D.C. Hayes Associates, Inc.

MICROCOMPUTER PRODUCTS

MICROMODEM II™

For the APPLE II* Personal Computing System



MICROMODEM II OWNER'S MANUAL

Written by Donald J. Hyde

Second Edition, May, 1979

NOTE:

Printed in U.S.A.

MICROMODEM II OWNER'S MANUAL

TABLE OF CONTENTS

INTRODUCTION

The D.C. Hayes Associates, Inc. Micromodem II, with its Microcoupler forms a complete low-speed data communications subsystem for the Apple II computer. Its small package combines all of the functions normally needed to perform most common data communications functions. These include a modem which is compatible with the popular Bell System 103-type modem, the Microcoupler data access arrangement which allows you to connect the modem to the telephone line, and all the programs you need to make it work in an on-board read-only memory (ROM).

This manual tells you how to install and use this powerful system. Since many of its functions and features are due to the on-board firmware, much of the manual is devoted to describing its operation and use.

This is a user's manual. It is intended for people who are primarily interested in using the Micromodem II, so it contains background information, detailed instructions, hints, suggestions, and a detailed description of the functions of all programmable registers and controls. It does not contain much information about the construction or design of the equipment.

HARDWARE

The Micromodem II is contained on a single small printed circuit board which fits in one of the peripheral slots of the Apple II computer. Its main feature is a low-speed modem which is compatible with a Bell System 103-type modem. It is capable of operating in either originate or answer mode at 110 or 300 baud. Also on the card are circuits which make it possible to perform automatic answering and automatic dialing.

Probably most important of all to the user is the lk-byte ROM containing all the programs necessary to make the rest of the circuitry useful. Programs stored in a ROM in this fashion are often called firmware.

FIRMWARE

The Micromodem II ROM firmware supports three distinctly different operating modes. All three modes are designed to be as simple as possible to use, while retaining all the flexibility needed for the most complex applications.

Terminal Program

With a few keystrokes, you can activate the built-in terminal program which simulates the operation of a dumb CRT terminal. In this mode you can use your Apple II computer to call and communicate with any computer equipped with a Bell System 103-type compatible modem, such as a time-sharing service or another Apple II computer equipped with a Micromodem II.

A unique feature of this terminal program is that all its options and features are controlled by codes typed in from the keyboard. By typing a few simple codes, you can command your Micromodem II to dial the phone, hang up the phone, or change baud rates.

Remote Console

Any time that the Micromodem II has control of the Apple II keyboard, it is constantly checking to see if the phone is ringing. If the phone rings, the Micromodem II will answer it, and if the caller is using modem, he will get control of the Apple II keyboard.

So to use your Apple II like a time-sharing computer, all you need is a dumb terminal and a Bell System 103-type compatible modem. Of course, another Apple II computer with a Micromodem II is an ideal terminal for this purpose.

Operating under Program Control

A BASIC program in the Apple II can dial the phone, answer the phone, transmit data, receive data, and hang up the phone. All of these functions are accomplished easily with regular BASIC INPUT and PRINT statements. PEEK's, POKE's, and machine-language subroutines are not needed for most applications.

But if you do need to do something more esoteric, all the modem and telephone functions can be controlled with a few PEEK'S and POKE's.

With this programmable power, you can write BASIC programs that can:

*dial your friends phone numbers when you type their names
*disseminate information to anyone who calls with a modem
*call a big time-sharing computer for help with large problems
*access a large data base for information such as stock market reports

And with a few other peripherals, your Apple II can:
*turn off the lights on command from a distant terminal
*call another computer for help if the basement floods

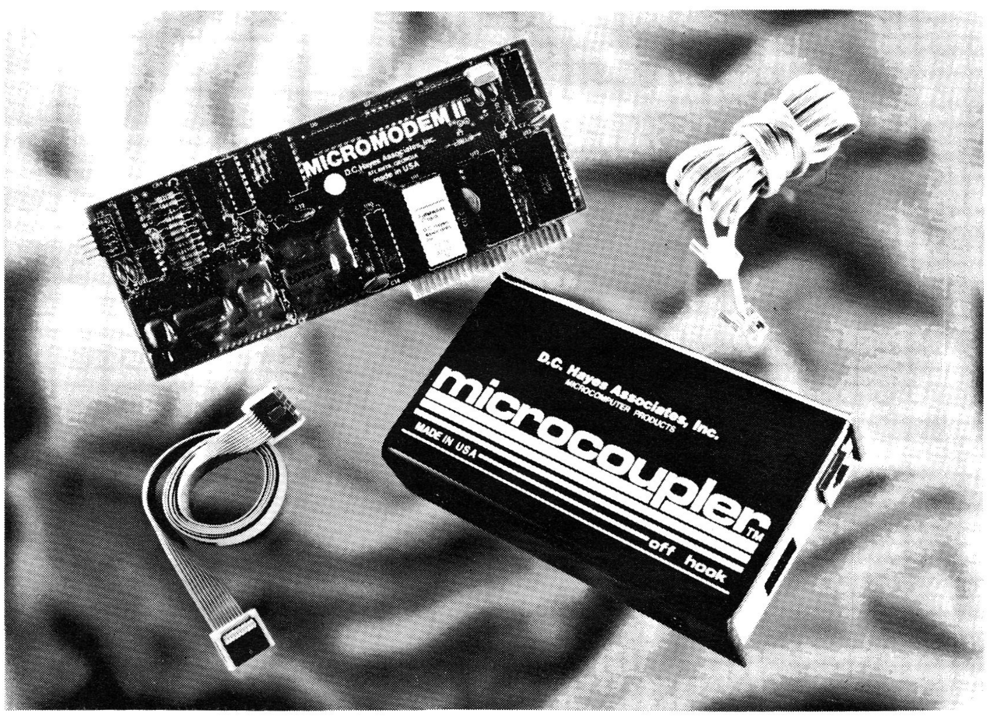
INSTALLATION

Identifying the Parts

Your Micromodem II is packed with several necessary items. You should take them out of the box and verify that they are all present and in good condition. They are:

1) Micromodem II printed circuit board.
2) Microcoupler for attachment to the telephone line.
3) Ribbon cable to connect Micromodem II to Microcoupler.
4) Modular telephone cable to connect Microcoupler to telephone line.

fig 1



Installing the Micromodem II in the Computer

!!!
------- MAKE SURE THE COMPUTER IS TURNED OFF! -------
!!!

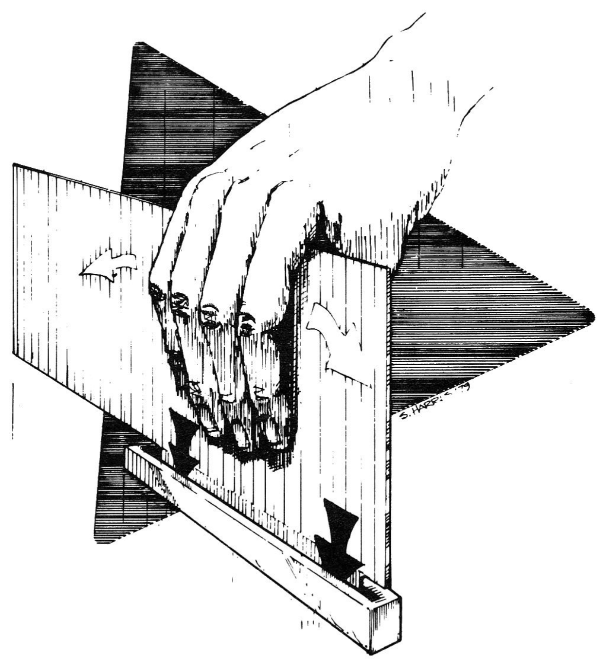
Remove the cover from your computer. Along the back wall of the case, you will see a row of eight rectangular printed circuit edge connectors. They are numbered 0 to 7 from left to right. If you look closely, you will see that they are marked with their numbers on the edge of the main circuit board right next to the back wall.

The Micromodem II will work in any of these slots except slot 0, the leftmost one. This socket is reserved for special functions such as extended BASIC firmware. Slots 6 and 7 are reserved for disk controllers, so although the Micromodem II will work there, it is best to save those slots.

<u>Are you sure the computer is turned off?</u>

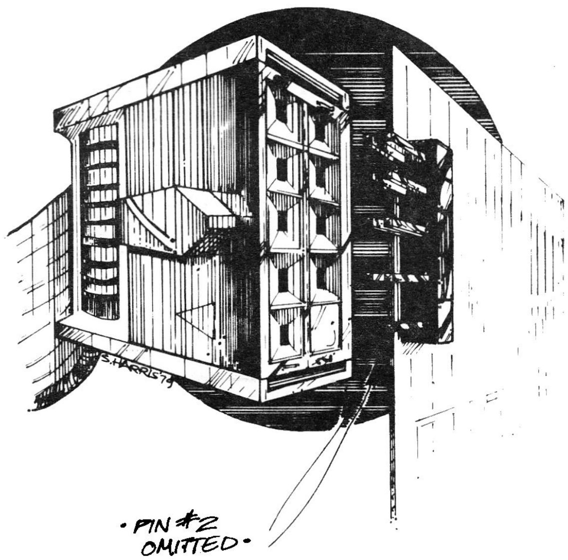
Once you have chosen a slot, insert the gold-plated fingers of the Micromodem II circuit board into a slot and press it in firmly. It helps to rock the board back and forth slightly. Make sure that it is firmly seated all the way into the socket.

fig 2



On the end of the circuit board toward the keyboard, you will see a

connector which consists of group of small pointed prongs. This is the connector for the ribbon cable which connects the Micromodem II to the Microcoupler. If you look closely, you will see that one of the prongs has been cut off. Look at the connector on the end of the ribbon cable. One of the holes in the connector is filled with a white plastic block. The filled hole matches the missing pin and forms a key which makes it difficult to plug it in wrong.



Both ends of the ribbon cable are the same, and are interchangeable. Pick one end and plug it into the Micromodem II board. If you have any difficulty plugging it in, try turning it over -- it might be backward. When the connector is properly mated, all of the pins will fit easily into their corresponding holes, and the connector body will be flush with the printed circuit board.

Run the ribbon cable out through one of the cable slots in the back wall of the case. You may now replace the cover.

The other end of the ribbon cable connects to the Microcoupler. On one end of the Microcoupler you will see two connectors. In the center of the end wall is a rectangular hole about an inch long. Just peeking through this hole you will see another set of pins just like the ones on the Micromodem II board. They are also keyed in the same fashion.

Plug the remaining end of the ribbon cable into this connector. Again, if you have any difficulty, check that it is turned the right way.

You are now ready to connect your computer to the national telephone network.

CONNECTING TO THE TELEPHONE NETWORK

There are a few details that we need to take care of now before you can legally connect your computer to the telephones.

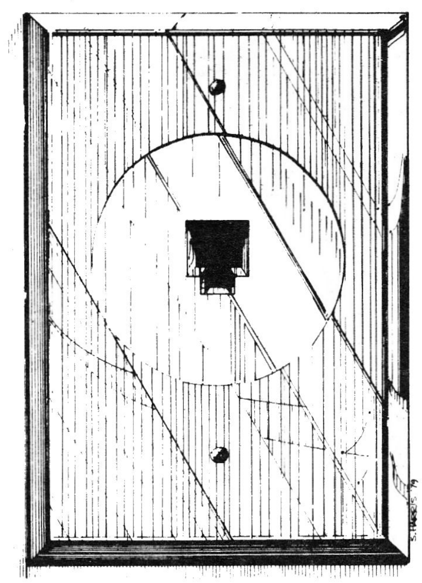
Legal and Technical Details

First you CANNOT legally connect your computer to either a party line or a pay phone.

Next you need a place to plug it in. If your home or office is already equipped with modular telephone jacks, then you already have this. You can recognize a modular telephone jack by the small squarish hole (about 3/8") where the modular plug goes in. If you have jacks with four round holes about 3/4" inch apart, then you don't have modular jacks, but an older type of connector.

A four-prong to modular adapter plug is available from your computer dealer.

fig 4



If you don't have modular jacks, your telephone company will be glad to install them for you. They will charge you for a service call (roughly $35.00 in Atlanta) plus a small fee for each jack installed (about $3.00 in Atlanta). If you have them installed, you should have several put in so that in the future you will be able to plug in phones and all the other new telephone goodies like modems and answering machines.

Modular jacks come in two varieties, a style which fits into a standard electrical box such as the ones that light switches and electrical outlets are mounted in, and a style which is mounted in a small plastic box which can be

screwed onto a wall or baseboard. Your phone company will probably know what you mean if you ask for modular plugs, but the official telephone company Uniform Service Order Code or USOC for the electrical box version is RJ11W and for the baseboard-mounted version it is RJ11C. Use these numbers when you order and there should be less possibility for confusion.

If you want to plug your computer into a jack that already has a telephone in it (and you don't want to have to unplug the telephone), you can get a Y adapter which has one male plug on one side and two female jacks on the other side.

Before you plug anything into the telephone line you must notify the telephone company. One reason this is necessary that they normally test all their lines on a regular basis. To do this they need to know what's attached to each line, and how many ringers there are on the line. The test measures how much current is drawn on the line. If it draws too much or too little, then they will know that there is something broken or that someone has been tampering with the line. This is how they find illegal extension phones.

You should call the telephone company business office (their telephone number is probably printed on your phone bill) and tell them that you are preparing to install an FCC registered device and wish to notify them. They will need to know what line you are connecting it to (the phone number), the FCC registration number (BI986H-62226-PC-E), and the ringer equivalence number (0.4B). If you plan to move your computer around, you are allowed to give the phone company a list of phone numbers. Then you won't have to notify them again as long as you move it to one of the lines on your list.

If your phone company has any additional questions about the Microcoupler, tell them to call us in Atlanta.

You are also required to notify the phone company whenever you permanently remove the Microcoupler.

If you experience trouble with your telephone line, you must first disconnect the Microcoupler and make sure that it is not causing the problem. The phone company cannot be responsible for problems caused by your equipment connected to their lines.

In the unlikely event that your Microcoupler causes trouble with your telephone line, you must disconnect it and not use it until it has been repaired. In order to keep FCC approval in force (and thereby be legal), all repairs must be performed by D.C. Hayes Associates, Inc. or our authorized agents. We have a very efficient repair department which seldom has any work to do, so we can repair any problem with your Microcoupler or your Micromodem II very promptly. Please see your warranty for full details.

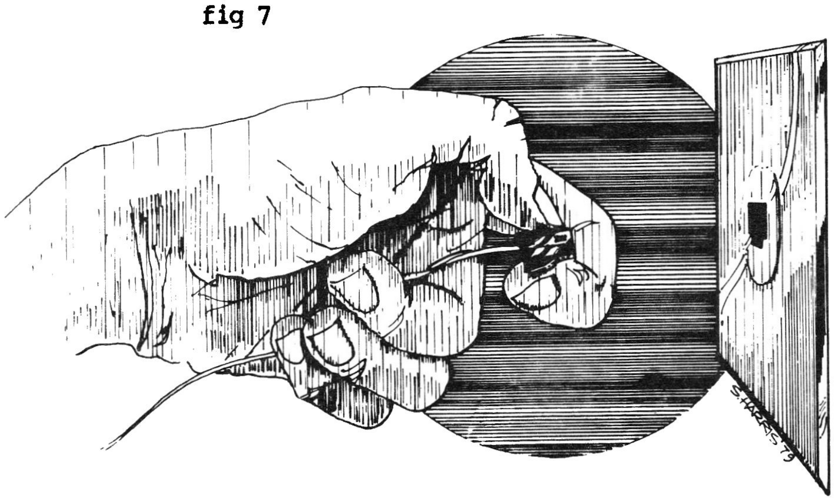
One more legal technicality and you can plug it in. We are obliged to inform you that the telephone company has the right to change their equipment and is under no obligation to make sure that their new equipment is compatible with your Microcoupler. It is very likely that they will change their equipment, but it is extremely unlikely that those changes will render your

Micromodem II or your Microcoupler unusable. The interface between the Microcoupler and the telephone line is functionally identical to the interface between a standard dial phone and the telephone line. Since the phone companies have roughly 100,000,000 dial phones operating today, they are very unlikely to install any new equipment that will render them all obsolete. It would cost too much to replace all those telephones.

Plugging in to the Telephone Line

Pick up the modular telephone cable. You will see that there is a small plug on each end. The plug is molded of clear plastic and has 4 gold contacts on one side and a small plastic tab on the other. The plastic tab forms a latch which will hold the plug in the jack. Insert one of the plugs into the modular jack on the Microcoupler. It will only go in one way, and when you put it all the way in, it will make a small snap as the latch takes hold. Pull on the cord. It will not unplug unless you press on the plastic tab.

fig 7



Now you can plug the other end of the modular cable into the telephone jack on the wall.

This completes the installation of your Micromodem II and you are now ready to try it out. The next chapter will show you how to use the built-in terminal simulation program to call any time-sharing system or other computer system.

Using the Micromodem II's built-in self-test capability

The Micromodem II hardware has a self-test capability built in. This feature is controlled by a bit in the modem control register. When this feature

is selected, the modem transmitter and receiver both operate on the same frequency band so that the receiver can receive the data sent by the transmitter. The Micromodem II hardware is designed so that when it is disconnected from the Microcoupler, there is a calibrated "leak" from the output to the input. This leak simulates the losses which would be encountered on a typical phone connection. Using this capability, a program can actually test virtually all of the hardware on the Micromodem II circuit board without even being connected to a phone line.

The selftest program below is written in integer BASIC, and will run in any Apple II configuration. It does not use any IN# or PR# statements, so it will run under D.O.S. without modification. The program utilizes a special entry point in the Micromodem II firmware which allows it to get around a "false read" problem which would otherwise be encountered due to certain peculiarities of the 6502 microprocessor.

The program tests the Micromodem II hardware by setting it up in each of the four possible combinations of mode and baud rate, sending all 128 valid ASCII characters through the modem, and verifying that they are all received correctly. Most of the testing is performed by two subroutines; at 10000, which performs the test and at 20000, which sets up the modem for the next test.

Lines 200-425 assign names to the various memory addresses used by the test routines. The tests are called by lines 700 and 900. Line 1000 adds the number of errors in each test to the total errors, which is reported at the end of the test by lines 1250-1400.

The test subroutine starts by clearing the ACIA and setting it up in normal mode in lines 10300 and 10400. The number of characters sent and received are set to 0 in line 10450.

Lines 10500-10600 form a loop in which the Receiver Register Full and Transmitter Register Empty bits are checked in the ACIA status register. When a character is present in the receiver, line 10550 detects this and goes to line 11100 where the character is read and checked for correctness. Whenever the transmitter register is empty, line 10600 drops through to lines 10700-11000 which send the next character unless all 128 have aleady been sent.

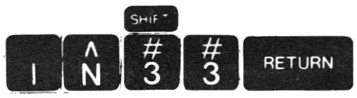
Lines 10800 and 10850 use a special entry point in the Micromodem II firmware to send each byte. Line 10900 bumps the count of characters sent, which also forms the next character to send.

Line 11100 reads the character which has been received, and line 11200 checks it. If it is incorrect, line 11300 counts the error, then line 11400 increments the received character count.

Line 20000 turns off the modem carrier, and 20050 resets the ACIA. The program waits at line 20100 until the carrier detect circuit responds (it has a built-in delay of about 1/2 second).

Lines 20150-20270 tell the operator what's happening, then line 20300 turns the modem on in the next mode to be tested. The next set of test conditions is

From Apple II BASIC:



Unless you are using a printer as described at the end of this section, you should also have a PR#0 statement in effect when you are using the terminal mode. If you have a PR#3 statment in effect when you enter terminal mode, you may find that your characters are being doubled at the other end of the line, or other strange things may happen.

ATTENTION!

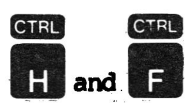
All the commands from the keyboard begin with an "attention" code:



When the Micromodem II firmware recognizes this code, it displays on the screen:

MICROMODEM II:?

Once you have its attention, you may type any of 8 command codes which control the operation of the terminal program and the modem.



These are the commands that start the terminal program.



starts the terminal in half-duplex mode (see BACKGROUND (p. 60) for an explanation of this term), and



starts it in full-duplex mode. When you type either of these commands, the Micromodem II displays:

MICROMODEM II:BEGIN TERM

If you are calling another Apple II equipped with a Micromodem II set up in remote console mode, you must use full duplex. If you want to communicate with another Apple II which is in teminal mode, both computers should operate half-duplex.

CTRL X

This command tells the Micromodem II to exit terminal mode. This re-establishes communication between the keyboard and programs (such as the monitor) in the Apple II . When you type this command, the Micromodem II displays:

MICROMODEM II:END TERM

When you exit from terminal mode, whatever program you might have left will still be EXACTLY where you left it; i.e. if you left a BASIC program which was waiting for you to type something, it will still be waiting.

CTRL Q

This command is accepted only in terminal mode when the telephone is hung up. It instructs the Micromodem II to pick up the phone and start dialing. When you type it, the Micromodem II responds:

MICROMODEM II:DIALING:

The Micromodem II then picks up the phone to start dialing. The flashing cursor disappears from the screen for two seconds while the Micromodem II waits for a dial tone. When it reappears you may start typing in the digits of the phone number. Each digit is dialed as you type it, and while it is being dialed, the cursor again disappears. It reappears when the Micromodem II is ready for another digit. Since the keyboard buffers one character, you may type the next digit as soon as the previous one appears on the screen.

The Micromodem II accepts and dials the digits 0 thru 9. An asterisk (*) instructs the Micromodem II to delay 2 seconds. This is useful for dialing through a PBX where it is necessary to wait for a second dial tone.

You can see the digits being dialed if you look at the off-hook LED on the Microcoupler. You can also hear the faint clicking of the off-hook relay in the Microcoupler. Don't try to pick up another extension and hear the Microcoupler

dialing, though, because the second phone will prevent the dial pulses from being recognized by the telephone exchange.

When you have typed in all the digits, type RETURN and the Micromodem II will begin listening for another modem to answer and turn on its carrier. If there is no carrier after 30 seconds, the Micromodem II will display:

MICROMODEM II:NO CARR.

MICROMODEM II:HUNG UP

You can try again by typing:

CTRL A CTRL Q

and retyping the phone number.

When the Micromodem II does hear a carrier, it displays:

MICROMODEM II:CONN.

At this point you have a connection with whatever has answered the phone, and you and it may begin exchanging data. Anything you type is transmitted to it, and anything it transmits is displayed on your screen.

Any time after you finish dialing, you can pick up on another extension and listen to the line. This is a good idea if you have tried dialing and not gotten through. You might find that your friend's computer is not on the line and a person has answered, or that you got a busy signal or a wrong number.

Once the modems have started their tones, you should hang up, because any noise you make will interfere with the modem signals. And the second telephone on the line reduces the strength of the signal, which increases the probability of errors.

CTRL Z

This command tells the Micromodem II to hang up the phone. While you are dialing and when the Micromodem II is waiting for a carrier tone, you do not need to precede this command with a

CTRL A

(you already have its attention). When you type this command, the Micromodem II

responds:

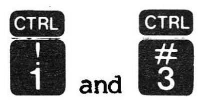
MICROMODEM II:HUNG UP

If at any time during a call, the carrier tone disappears for a half second or more, the Micromodem II will display:

MICROMODEM II:NO CARR.

MICROMODEM II:HUNG UP

And it will hang up the phone. This is done so that if the line goes down or the modem at the other end hangs up, the Micromodem II will hang up too rather than hold the line so that it will be busy.



These codes change the baud rate of the Micromodem II.



sets the baud rate to 110 baud (with 2 stop bits), which is needed to communicate with a model 33 or a model 35 Teletype.



sets the baud rate to 300 baud (with 1 stop bit), which is preferable in most cases because it is 3 times faster. Sometimes if you are having difficulty with a poor phone connection, the slower baud rate may be helpful because it is slightly less susceptible to errors.

These commands do not have any displays.



This command simulates the effect of holding down the break key which is found on many terminals. On some time-sharing systems, this function is used to stop unwanted output (somewhat like

CTRL C for an Apple BASIC program). The break condition remains in effect until you type any other character.

This command does not have any display.

USING A PRINTER IN TERMINAL MODE

If you have a printer on your Apple II, it is possible to cause the output in terminal mode to be sent to the printer instead of to the display. In order to do this, simply select the printer for output just as you normally would before entering terminal mode in the Micromodem II.

Depending on which printer interface and what printer you are using, this may or may not work. In order to avoid losing characters that are arriving down the telephone line at a steady rate of 10 or 30 per second, it is essential that the printer be able to print them AT LEAST that fast.

If your printer is connected via a parallel printer interface card, it will most likely buffer a whole line at a time and then print the whole line after it receives a carriage return. Since putting characters into a buffer is very fast, there is little likelyhood that any of the characters will be missed. BUT, when the carriage return comes down the line, the printer will go to work. Printing a whole line can take a fair amount of time (a second or more), during which time you will have missed quite a few characters on the next line.

Most timesharing computers send a few rubout or null characters at the end of a line to allow enough time for printers to return their printheads to the left margin, but this delay is generally only a few tenths of a second at most.

If you are dialing into an Apple II with a Micromodem II, you can select a delay after carriage return of up to 2.55 seconds, which is adequate for most printers.

If your printer is connected via a high-speed serial interface card, all that is necessary is that the baud rate of the printer be at least as great as the baud rate you are using on the modem. If the baud rates are equal, it is essential that the character format also be the same. Please note that the high-speed serial card defaults to a character format with 2 stop bits, and that at 300 baud the Micromodem II (and most of the systems you might call with it) defaults to 1 stop bit. The additional stop bit will make the printer about 10% slower than the modem and will cause about 1 character out of 10 to be lost.

EXAMPLE SESSION

We have just finished installing our new Micromodem II in slot 3 and are going to use it to call the XYZ time-sharing corp's BIG9999 computer system.

First we turn on our computer and press:

SHIFT
RESET ESC P RETURN

to clear the screen and bring up the Apple II monitor's prompt. Our screen looks like:

*

To connect the keyboard input of the Apple II to the Micromodem II, we type:

CTRL
3 K RETURN

Now we have another * from the monitor. To start the terminal program running, we then type:

CTRL CTRL
A F

We knew beforehand that the BIG9999 operated in full duplex mode (most timesharing machines do) which is why we typed

CTRL F instead of CTRL H

If you do not know whether the system you are calling is full or half duplex, it's easier to assume half-duplex to begin with. If you are wrong, each character you type to the time-sharing machine will appear twice on your screen. You can type:

CTRL CTRL
A F

any time and it will change the mode of the terminal program in your computer. None of the commands are sent to the time-sharing machine.

We also knew beforehand that the BIG9999 was set up to run at 300 baud. Since this is the speed the Micromodem II assumes as a default value, we didn't need to type any commands to set the baud rate.

At this point our screen shows:

MICROMODEM II:?

MICROMODEM II:BEGIN TERM

Then we type:

CTRL A CTRL Q

to tell the Micromodem II we want to dial a telephone number. The Micromodem II responds:

MICROMODEM II:?

MICROMODEM II:DIALING:

At this point the Micromodem II has picked up its telephone. It waits 2 seconds for a dial tone, and then puts up a cursor right after the word DIALING:. We type in the phone number, one digit at a time. The cursor reappears whenever the Micromodem II is ready to dial another digit. To make the display more readable, we type parentheses and dashes at the appropriate places. They are ignored by the Micromodem II.

When we have typed in all the digits of the phone number, we press:

RETURN

Then we look over the number we've dialed to make sure it's right. If we made a mistake, we can type:

CTRL Z

and the Micromodem II will hang up so we can try again. So now the display shows:

MICROMODEM II:DIALING:1(404)555-1212
MICROMODEM II:AWAIT CARR.

The Micromodem II will wait up to 30 seconds for a modem to answer the phone and send its carrier. Since we are lucky and the moon is full, the BIG9999 is up and answers on the first ring. The display now shows:

MICROMODEM II:DIALING:1(404)555-1212
MICROMODEM II:AWAIT CARR.

MICROMODEM II:CONN.

HELLO THIS IS THE XYZ CORP BIG9999 TIME-SHARING SYSTEM.

PLEASE SIGN ON:

Now we sign on and work with the time-sharing machine. As often happens, something goes wrong and the BIG9999 crashes. We wait a few minutes to see if it will recover, but soon lose patience and decide we want to play with our own computer. So we type:

CTRL CTRL
A Z

and the Micromodem II hangs up. Now our display shows:

```
BLAH BLAH BLAH .
                .
                 ..
                  .
                    .
                     .
                      .
                 ********
              **  ****  *****          <--- (BIG9999 crashing)
                **  ****** *
                     **
                    BOOM!
```

MICROMODEM II:?

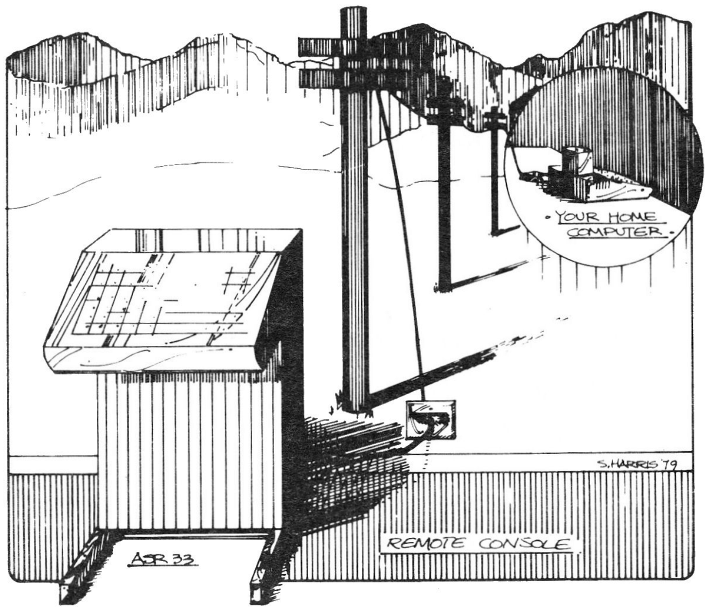
MICROMODEM II:HUNG UP

We could also have simply pressed

RESET

which would also have disconnected the Micromodem II from the Apple II's keyboard and hung up the phone.

REMOTE CONSOLE



It is extremely easy to set up the Micromodem II so that it will answer the phone and allow you to call in with any terminal and use your Apple II computer. In this example, I am assuming for simplicity that your Micromodem II is in slot 3. If yours is in another slot, substitute the proper slot number where 3's appear. If you are in the monitor, simply type:

3 CTRL-K RETURN

3 CTRL-P RETURN

If you are in BASIC, then type:

I N SHIFT-# 3 RETURN

P R SHIFT-# 3 RETURN

These commands cause the Apple II keyboard to be routed through the firmware on the Micromodem II card. Whenever any program in the Apple II is requesting input from the keyboard (which is most of the time), the Micromodem II firmware will check to see if the phone is ringing. When the phone rings, it displays:

MICROMODEM II:RING

It then waits until the end of a ring and answers the phone. At this point it displays:

MICROMODEM II:AWAIT CARR.

It then turns on its carrier and waits up to 30 seconds for a modem at the other end of the line to respond with its carrier. When it detects this carrier, it displays:

MICROMODEM II:CONN.

When it connects, the Micromodem II sends a

RETURN

to the Apple II's input. The RETURN will cause the Apple II to send out its prompt so that the caller can tell whether the Apple II is in BASIC or in the monitor.

The Apple II is now under remote control. Almost anything that you could do from the Apple II's own keyboard you can do from the remote terminal. If you are sitting next to the Apple II, you can see everything that is going on, because all output to the modem also appears on the screen. The local keyboard is also active and anything you type on it will be accepted as input to the Apple II.

Special Control Characters

There are six special control characters which are accepted ONLY FROM A REMOTE TERMINAL. These control characters may be disabled by setting the code transparency bit (TRAN) in the FLAG word.

CTRL
S

This code (stop) tells the Micromodem II to temporarily stop sending output. This is useful if you want to stop to look at some output which is scrolling by too fast for you to read. It is also valuable if you are sending

to another computer or to some printers which are designed to send this code when their buffers fill up.

Sending any other code than control S will restart output. Sending another control S will cause one more character to be sent.

CTRL Y

This control code has the same effect (almost) as pressing the RESET button on the Apple II's keyboard. The difference is that it does not produce a hardware reset pulse (which would reset the Micromodem II and make it hang up), and it does not return the Apple II's input and output to the keyboard and display (which would disconnect the Micromodem II and hence the remote terminal).

CTRL R

This code will get the Apple II out of the terminal program if it should have been in that program when you called. With this command, you can "take over" control of an Apple II computer which you are connected to and use all of its facilities, or communicate with a program which is running in it.

CTRL T

This command is the reverse of control R. It re-enables the Apple II's keyboard and starts the terminal program running. It could be useful if you wanted to type back and forth with someone sitting next to the Apple II. Any data or programs in the Apple II will be undisturbed until you type control R from the remote terminal or the person with the Apple II types control A control X to stop the terminal program.

CTRL Z

This tells the Micromodem II to hang up the phone. Nothing else will be changed. All programs and data in the Apple II will remain undisturbed. If you call back again, the Micromodem II will answer the phone and you will be right back where you were.

The Micromodem II will also display:

MICROMODEM II:HUNG UP

CTRL ^N

This command sets up the Micromodem II for communications with a printing terminal. It does several things. First it enables the insertion of a line feed after carriage returns. Without this, most printing terminals (and many CRT terminals) will print over and over again on the same line. It also enables a short delay after the line feed to allow time for the physical movement of the print head. This delay is initially set to 30 msec. This may be changed by setting location CRDLY ($5F8+N, Decimal 1528+N). This delay is in increments of 10 msec. i.e. a value of 3 specifies a 30 msec. delay. It also clears the Apple II's screen and disables the local display. This effectively disables the Apple II's internal line-folding algorithm so that you will be able to use the full width of your terminal.

Discard Characters

On input, the Micromodem II automatically discards three characters which could interfere with proper operation of the Apple II's firmware. These are NULL (all zeroes), RUBOUT (all ones) and LINE FEED. This feature is also disabled if you select the code transparency option.

Hanging up on the Micromodem II

If you should hang up the phone while you are dialed in to the Micromodem II, it will detect the loss of carrier and hang up. This means that you will be able to dial back in and it will answer again so you can get back to where you were. The Micromodem II displays:

MICROMODEM II:NO CARR.

MICROMODEM II:HUNG UP

Note on Cursor Movement

When you are using the Apple II computer from its own keyboard, there are several cursor movement commands that are very useful for editing programs or other material which may be displayed on the screen. Some, but not all, of

these commands may also be used from a remote terminal.

The Apple II keyboard left-arrow (ASCII BS, Ctrl-H) character causes the Apple II to move its cursor left one position on the screen and causes it to discard the last character from its input buffer. This code is as valid from a remote terminal as from the keyboard, but your terminal will not necessarily backspace, since this cursor movement is a function of the Apple II's display. The Apple II firmware does echo this character, and many CRT terminals do recognize this standard ASCII character as a backspace.

The Apple II keyboard right-arrow (ASCII NAK, Ctrl-U) character causes the Apple II to move its cursor right one position and take the character under the cursor on the screen as its input. This sort of works from a remote terminal, if the Apple II's local display is enabled. If the local display is not enabled, then the cursor will not move and it will always pick up the same character. If the display is enabled, The Apple II will pick up the next character from the screen, but your terminal will not move its cusror unless it recognizes the ASCII character ACK as a command to move its cursor forward. Since this is not a standard ASCII character function, it will not be likely to work on your terminal.

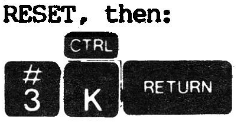
The four escape sequences which cause cursor movement on the Apple II work, but are pretty much useless unless you are able to see the Apple II's screen. Since these codes are not echoed by the Apple II's firmware, it would not be possible for your terminal to move its cursor the same way even if it were designed to recognize these character sequences.

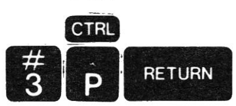
Hint for D.O.S. Users

When D.O.S. is booted, it executes an IN#0 and a PR#0 statement. If you are connected via a Micromodem II, these statements will effectively disconnect you. You can get around this problem by adding appropriate IN# and PR# statements to the "hello" program that D.O.S. executes when it is booted.

EXAMPLE

Before leaving for work in the morning, we turn on our computer, press RESET, then:





At lunch time, we go into the terminal room at the office and borrow one of the terminals. We then dial up our home number and the Apple II answers. The terminal shows:

```
 FF FF FF FF FF
*
```

The row of FF's above the * prompt is a portion of a memory dump printed by the Apple II monitor program. It is there because the Apple II's monitor was given a RETURN when the Micromodem II answered the phone. This helps you to verify that you did reach your Apple II. The * is the Apple II monitor's propmt.

We then type:

CTRL B RETURN

We now receive the Apple II BASIC prompt > which tells us that the Apple II is ready to run BASIC. We then type in the BASIC program we have been thinking about all morning while we were supposed to be working. It sure beats working, but all too soon lunch hour is over, so we hang up and go back to work.

That evening when we get home, we go to our computer and press:

RESET CTRL C RETURN

This disconnects the Micromodem II and gets us back into BASIC. The BASIC program we typed in at lunch is still there and we have had a great idea to improve it. So we put in our new idea and try it. Then we save the program on tape or disk.

PROGRAMMING THE MICROMODEM II

The Micromodem II and its built-in firmware are designed to be easy to use from BASIC programs in the Apple II. Most of the commonly needed functions can be performed with the ordinary BASIC INPUT, PRINT, IN#, and PR# statements. Dialing and hanging up the phone require the use of a couple of control characters, but it is quite simple to put these characters into strings in the BASIC program.

There are several more advanced functions such as nonstandard data formats and code transparent operation which will require a few PEEK and POKE statements. In fact, it is possible to access all the Micromodem II's hardware features through PEEK and POKE statements.

LANGUAGE NOTE

All of the examples in this section were written in INTEGER BASIC. Most of them will also work unchanged in APPLESOFT II, but they do not take advantage of some of APPLESOFT II's features, in particular the CHR$ function, which can make programs much more readable.

D.O.S. NOTE

These examples contain many IN# and PR# statements. To work with D.O.S., you will need to modify those statements as described in the D.O.S. manual.

Elementary BASIC Programming

There are five primary operations that programs are most likely to need to perform in order to use the Micromodem II. These are:

1) Dial the telephone
2) Hang up the telephone
3) Wait for the telephone to ring and answer it
4) Transmit data via the Micromodem II
5) Receive data via the Micromodem II

Dialing the Telephone

To dial the telephone, we must select the Micromodem II for output and send it a control Q followed by a phone number in ASCII followed by a RETURN. Assuming (as before) that the Micromodem II is in slot 3, the following program

fragment will dial long distance information for the Atlanta area:

```
100 DIAL$=""
200 PR#3:PRINT DIAL$;"1(404)555-1212"
300 END
```

Note that the empty-looking quotes in statement 100 are not really empty. They contain a control Q, which does not show on the listing because it is a non-printing character. You can type control characters into quoted strings and they will be there (you can verify this with the LEN function) even though you cannot see them. If you are using APPLESOFT II, you can make your programs more readable by using the CHR$ function. In APPLESOFT II, you can replace line 100 with:

```
100 DIAL$=CHR$(17)
```

Statement 200 does all the work. First it selects the Micromodem II for output (PR#3), then it sends it a control Q followed by the phone number. The RETURN is automatically supplied by BASIC at the end of any PRINT statement unless the statement ends with a semicolon (;).

When statement 200 is executed, the Micromodem II will display:

```
MICROMODEM II:DIALING:1(404)555-1212
MICROMODEM II:AWAITING CARR.
```

Once it has dialed the telephone, the Micromodem II will begin waiting for a carrier. It will wait up to 30 seconds before it gives up. Statement 300 will not be executed until either the 30 seconds has elapsed or the Micromodem II has detected a carrier. If it gives up, it will display:

```
MICROMODEM II:NO CARR.

MICROMODEM II:HUNG UP
```

If it detects a carrier it will display:

```
MICROMODEM II:CONNECT
```

Since it is possible for the dialing to be unsuccessful, the program should check to be sure that a carrier was detected. The following program fragment will type "GOT IT" if a carrier was detected and "NOT HOME" if no carrier is detected.

```
400 IF PEEK(1656+3)>127 THEN GOTO 700
500 PRINT "NOT HOME"
600 END
700 PRINT "GOT IT"
800 END
```

Memory location 1656 + <slot number> contains the modem control word. The

most significant bit of this word (which has a binary weight of 128) controls the telephone switch hook. This bit is one any time the Micromodem II has the phone off the hook. This memory location and several others containing Micromodem II status bits of several kinds are described in more detail in ADVANCED PROGRAMMING (p. 33).

An asterisk character (*) in a phone number causes the Micromodem II to delay for 2 seconds. You may find this useful if you connect your Mcromodem II to a PBX (private switchboard) in which it is necessary to dial a number to get an outside line. The delay allows time for the outside dial tone.

If the last character of the phone number is a control-J (LINE FEED) character, the Micromodem II will skip its usual sequence of listening for a carrier tone. This can be useful if you want your computer to dial the phone for some other purpose than to establish communications with another computer. Using this feature, you can program your Apple II to work as a repertory dialer.

Hanging up the phone

To tell the Micromodem II to hang up the phone, all the BASIC program needs to do is to output a control Z to it. The following program fragment will hang up the phone.

```
1000 BYE$=""
1100 PR#3:PRINT BYE$
1200 END
```

As in the first example, the empty-looking quotes really contain a non-printing control character. In this case it is a control Z. When statement 1100 is executed, the Micromodem II will display:

```
MICROMODEM II:HUNG UP
```

Answering the phone

To wait for the phone to ring and answer it when it rings, all a BASIC program needs to do is select the Micromodem II for input and perform an INPUT statement. The following fragment will wait for the phone to ring, answer the phone, wait for a carrier, and then transmit a short message identifying itself:

```
2000 IN#3:INPUT I$
2100 PR#3:PRINT "HELLO, THIS IS AN EXAMPLE PROGRAM"
2200 END
```

There will be no outward sign that anything is happening when statement 2000 is executed until the phone rings. When it rings, the Micromodem II will perform its usual phone-answering chores and will display:

```
MICROMODEM II:RING

MICROMODEM II:AWAIT CARR.
```

It will wait up to 30 seconds for a carrier. If it does not detect one, it will display:

MICROMODEM II:NO CARR.

MICROMODEM II:HUNG UP

If it does detect a carrier, it will display:

MICROMODEM II:CONNECT

When a carrier is detected and only when a carrier is detected, the Micromodem II will send a RETURN to the Apple II's input. This will satisfy the BASIC INPUT statement at line 2000 and allow the execution of the BASIC program to continue at line 2100.

The BASIC program will then transmit its message and stop at line 2200.

Transmitting Data via the Micromodem II

Once the Micromodem II has detected a carrier and established a connection, all the BASIC program needs to do to transmit data is to select the Micromodem II for output (if it has not aleady done so) and send the data using PRINT statements.

Receiving Data via the Micromodem II

To recieve data from a remote device via the Micromodem II, all that is needed is to select the Micromodem II for input (if it hasn't already been selected) and perform an INPUT statement.

EXAMPLE PROGRAM

The following example program was designed to illustrate the use of the concepts described here. It is a simple store-and-forward message switching program, which uses all the features described in this chapter.

It was written in INTEGER BASIC and will run on a minimal 4K Apple II computer. If you wish to use it in a D.O.S. system, you will need to modify all the IN# and PR# statements as described in the D.O.S. manual.

What the Example Program Does

The purpose of this program is to obtain a short message and store it in its memory for a period of time before forwarding it to another computer. This could be a useful function if the Apple II were connected to a WATS line or some other low-cost transmission facility. It could also save money by holding a message until late evening when the phone rates are their lowest.

When the phone rings, this program answers it and sends a short description

of itself. Then it asks for and verifies a secret password. If the password is not correct after 3 tries, it hangs up on the caller.

Then it requests the telephone number to which the message is to be forwarded, and inputs it. After explaining its limitations (5 lines of 40 characters), it inputs the message. Next it asks how long it should wait before forwarding the message and obtains a time. It then gives the caller a chance to verify all his input and start over if it is incorrect.

Once it has correct input, it tells the caller good bye and hangs up the phone. After waiting the specified length of time it dials the number supplied by the user. If it fails to detect a carrier, it hangs up, waits five minutes and tries again. It does this up to 3 times before giving up and restarting itself.

Once it has connected, it waits a few seconds for the machine it has called to finish identifying itself before sending the message it has stored. Once it has sent its message, it hangs up and goes back to waiting for the phone to ring.

How It Works

Line 90 makes sure that the variables used in the timing loop at 30000-30300 are declared at the beginning of the symbol table. This makes sure that the timing will not change if the program is changed.

Line 400 defines which slot the Micromodem II is plugged into. If your Micromodem II is not in slot 3 then you will need to change this line.

Line 1000 defines the secret password which is checked in line 2000. You will probably want to make up your own password. It is like the combination to a lock, and you should treat it as one. This program has a password because it is capable of making long distance phone calls and thus spending your money.

Line 1100 sets up the control codes for the Micromodem II and gives them convenient names. Line 1300 selects the Micromodem II for output and makes sure that the phone is hung up. Then the program waits for the phone to ring in line 1400. Notice that it conveniently displays on the screen what it is doing. Line 1500 checks the off-hook bit in the modem control word to see if a carrier has been detected. If there is no carrier, it goes back to waiting for a modem to call.

Lines 1800 thru 2500 solicit and check the password. Note that the number of tries is counted so that it can hang up on a caller who does not know the password.

Line 2900 gets the phone number, and lines 3000-3900 get the message. The waiting time is gotten in line 4200. Lines 4300 through 5500 show the caller his input and allow him to try again if it is not correct.

The program then hangs up in line 5700, waits T minutes in lines 5900-6100. It dials the phone in line 6300, checks for a valid modem connection in line

6400, and does retries in lines 6500-7000.

Lines 7375 through 7900 send the message, line 8000 hangs up, and line 8100 goes back to waiting for the phone to ring.

```
90 I=0:J=0:K=0
100 REM  SLOT MUST = SLOT MICROMODEM
    II IS IN
400 SLOT=3
500 DIM MES1$(40),MES2$(40),MES3$
    (40)
600 DIM MES4$(40),MES5$(40)
700 DIM NUMBER$(20),I$(40)
800 DIM PASS$(20)
900 REM  THIS SETS UP THE PASSWORD
1000 PASS$="SECRET PASSWORD"
1100 BYE$="":DIAL$=""
1300 PR#SLOT: PRINT BYE$
1400 IN#SLOT: INPUT "WAITING FOR THE
     PHONE TO RING",I$
1500 IF PEEK (1656+SLOT)<128 THEN
     GOTO 1300
1600 PRINT "HELLO, THIS IS A STORE AN
     D FORWARD"
1700 PRINT "PROGRAM RUNNING ON AN APP
     LE II"
1800 I=0
1900 INPUT "PASSWORD ",I$
2000 IF I$=PASS$ THEN GOTO 2500
2100 PRINT "INCORRECT PASSWORD"
2200 I=I+1
2300 IF I>2 THEN GOTO 1300
2400 GOTO 1900
2500 PRINT "CORRECT PASSWORD"
2600 PRINT "THIS PROGRAM WILL FORWARD
      A SHORT"
2700 PRINT "MESSAGE TO ANOTHER COMPUT
     ER OR TERMINAL"
2800 PRINT
2900 INPUT "PLEASE ENTER ITS PHONE NU
     MBER ",NUMBER$
3000 PRINT
3100 PRINT "YOU MAY ENTER A MESSAGE O
     F 5 LINES OF"
3200 PRINT "UP TO 40 CHARACTERS EACH.
     "
3300 PRINT
3400 INPUT "1)",MES1$
3500 INPUT "2)",MES2$
3600 INPUT "3)",MES3$
3700 INPUT "4)",MES4$
3800 INPUT "5)",MES5$
3900 PRINT
4000 PRINT "I CAN WAIT A FEW MINUTES
     BEFORE"
4100 PRINT "I FORWARD THE MESSAGE."
4200 INPUT "HOW MANY MINUTES, 0-32767
     ",T
4300 PRINT
4400 PRINT "THANK YOU"
4500 PRINT "IN ";T;" MINUTES, I WILL
     FORWARD THE"
4600 PRINT "FOLLOWING MESSAGE TO "
     ;NUMBER$
4700 PRINT
4800 PRINT MES1$
4900 PRINT MES2$
5000 PRINT MES3$
5100 PRINT MES4$
5200 PRINT MES5$
5300 PRINT
5400 INPUT "IS THAT OK?",I$
5500 IF I$(1,1)#"Y" THEN GOTO 2600
5600 PRINT "GOOD BYE"
5700 PRINT BYE$
5750 FOR I=1 TO 700: NEXT I
5800 REM  NOW WAIT T MINUTES
5850 IF T<1 THEN GOTO 6200
5900 FOR I=1 TO T
6000 GOSUB 30000
6100 NEXT I
6200 I=0
6300 PRINT DIAL$;NUMBER$
6400 IF PEEK (1656+SLOT)>127 THEN
     GOTO 7100
6500 I=I+1
6600 IF I>3 THEN GOTO 1300
6700 FOR L=1 TO 5
6800 GOSUB 30000
6900 NEXT L
7000 GOTO 6300
7100 REM  FIRST GIVE HIM SOME TIME
7200 REM  FOR AN ANSWERBACK MESSAGE
7350 FOR I=1 TO 2000: NEXT I
7375 PRINT "HELLO, MESSAGE FOLLOWS:"
7400 PRINT MES1$
7500 PRINT MES2$
7600 PRINT MES3$
7700 PRINT MES4$
7800 PRINT MES5$
7900 PRINT
8000 PRINT BYE$
8100 GOTO 1300
29900 REM  SUBROUTINE TO DELAY 1 MINUT
      E
30000 FOR J=1 TO 60
30100 FOR K=1 TO 700
30200 NEXT K,J
30300 RETURN
```

ADVANCED PROGRAMMING

This chapter describes concepts and design information needed for programming more advanced or sophisticated applications. First I will describe the hardware and the on-board firmware from a programmer's point of view. Then building on this knowledge, I will describe several useful techniques which you might find valuable in designing your own systems.

A Program's Eye View of the Micromodem II

As far as a program in the Apple II is concerned, the Micromodem II consists of 11 memory locations which have some special properties. These 11 locations are tabulated and described in the table at the end of the manual "Memory Locations Used By the Micromodem II" (p. 73). For additional details please consult the table.

Three of these locations correspond to six hardware registers on the Micromodem II circuit board. These locations are not like read/write memory locations because instead of a single read/write cell, each of these memory locations consists of a pair of cells, one read-only and the other write-only. The two cells in each pair are related to each other. In a few rare cases it is possible to read back what you have written to a location, but in general what you write to one of these locations and what you read back are two different things.

DATA, $C087+N0 = -16249+16*N dec.

One of the three pairs is for modem data. Data which is written to this location is transmitted through the modem, and data which is received from the modem is read from the same location. In self-test mode, data which is written to this location can be read back from the same location one character time later.

STATUS/CR1, $C086+N0 = -16250+16*N dec.

The second pair of cells reads the status and writes the controls of the Motorola 6850 ACIA chip. This chip performs all the parallel-to-serial conversion on output and serial-to-parallel conversion on input data. The bits of the status register each have separate meanings, and report on many important conditions in the chip. The most important of them are Receiver Register Full, which tells the program that there is a valid character present in the received data register (described above), and Transmitter Register Empty, which tells the program that it is ok to write another byte to the transmitter data register. Other bits report various errors which the ACIA chip is capable of detecting on received data.

RI/CR2, $C085+N0 = -16251+16*N dec.

The third location pairs modem controls and ring detect. Codes output to

this location control such functions as taking the phone off hook, turning on the modem transmitter, and setting its mode and baud rate. A program can deteremine whether the phone is ringing by reading this location.

CN, NO, CHAR

Three more locations are actual memory used by the Micromodem II firmware for temporary storage of variables. These locations are located in an area of the Apple II's memory which is set aside specifically for temporary variable storage by firmware on peripheral cards. These locations may be shared by other peripherals in your Apple II. The Micromodem II firmware is designed to follow a standard method of sharing this memory so that it will not interfere with other peripherals or be interfered with by them.

The Micromodem II firmware uses five other memory locations that are set aside specifically for the slot in which it resides. In order to be able to use these locations, the Micromodem II firmware is able to determine which slot it resides in. In order for your program to use these locations to communicate with the Micromodem II firmware, you will need to tell it what location the Micromodem II is plugged into.

MODEM, $678+N = 1656+N dec., and ACIA, 7F8+N = 2040+N dec.

Two of these locations are used to store the current contents of the two control registers described above. As you might recall, these registers are write-only, and if the firmware needs to know what it wrote to these registers before, it will have to remember what it wrote. It does.

If you intend to modify the settings of the two control registers, usually it is preferable to write to these two memory locations and then let the firmware take care of writing it to the actual hardware registers. It does this each time it transmits a byte of data to the modem, so all you need to do is POKE the byte you want into the appropriate memory location and then output one or more bytes with a PRINT statement.

LOCSE, $6F8+N = 1784+N dec.

This memory location contains a value which is exclusive-or'ed with all lower-case letters which are received through the modem. If this location contains the hex value 20, then all lower-case letters will be translated to the corresponding upper case letter. This location is initialized to hex 20 when the Micromodem II firmware initializes itself (the first time it is used, either for input or for output). If you write a 0 to this location, that will disable the lower-to-upper case translation, and lower-case letters will be passed through.

FLAGS, $778+N = 1912+N dec.

The fourth location contains flags which turn various firmware options on and off. Several of the flags are used internally by the Micromodem II firmware and if they are set by another program, the results are unpredictable. Four of them, however, are potentially useful and so are of interest to the programmer.

Bit 7, the most significant bit, controls the display of modem output. Normally all output to the Micromodem II is also displayed on the Apple II's display screen. This can be inhibited by setting this bit. In terminal mode, this bit determines the difference between half- and full-duplex. It is set for full-duplex, and reset for half-duplex.

Bit 1 controls whether the Micromodem II firmware accepts data from the local keyboard when it is selected for input. Normally data is accepted either from the modem input or from the Apple II's own keyboard. If this bit is 0, the local keyboard will be disabled as long as the Micromodem II is selected for input. Even when it is disabled, it will still respond to control-a and RESET.

Bit 2 selects code transparency. Normally the Micromodem II firmware responds to several control codes sent out by the program or received from the modem. In some cases this could interfere with your applications. This control bit allows you to turn these features off so that you can transmit and receive all 128 ASCII characters.

Bit 4 selects line feed insertion after carriage return. Normally, the Apple II firmware does not use line feed characters for any purpose, however most printing terminals and many CRT's require line feed characters to advance to the next line. Without line feeds, they print on the same line over and over again. This option also enables an adjustable delay after the line feed character. This delay is needed on most printing terminals to allow sufficient time for paper movement.

<u>CRDLY, $5F8+N = 1528+N dec.</u>

The remaining location holds the setting for the optional delay after a carriage return. The contents of this location specify the delay in increments of 10 milliseconds.

The Firmware

The firmware on the Micromodem II resides in a single 2708 ROM chip. It occupies two discontinuous areas of memory space in the Apple II. Each peripheral slot in the Apple II has 256 bytes of memory space allotted to it for firmware. The address of this space is determined by the slot number. Programs which occupy this space must be written so that they will work no matter what address they occupy, since the address varies depending on which slot they are plugged into.

The Apple II also allots a single 2048-byte area which can be shared by all peripheral boards in the machine for their firmware. This area always has the same address, which simplifies programming, but since it must be shared with all the other peripherals which may be in the machine, it has to have a bank switch. This switch turns the ROM in this area on when the Micromodem II is operating, and turns it off when the Micromodem II is inactive so that other peripherals may use the memory space. When the Micromodem II is operating, the entire 1024 bytes of the onboard ROM are mapped into the lower half of this space.

There is one location in the 256-byte slot-dependent area which you might need to access via a CALL statement. This is a special output data routine (located at CN02 hex = -16382+256*N decimal) which outputs the byte stored in location CHAR (778 hex = 1912 decimal) through the modem. A special routine is needed in some applications which are operating in full-duplex (such as the self-test program), due to an incompatibility between the 6502 microprocessor's indexed write timing and the 6850 ACIA chip. The BASIC POKE statement uses an indexed write, and a POKE to the DATA location will cause any data which the ACIA has received to be discarded. For an example of the use of this entry, see the self-test program lines 10800 and 10850 (p. 8).

MORE ADVANCED TECHNIQUES

Assumptions Made in the Examples

The examples in the following discussions all assume for convenience that:

1) The variable SLOT has previously been initialized to the slot number of the Micromodem II.
2) The system is not running under D.O.S. i.e. the simple form of the IN# and PR# statements is acceptable.

Changing Baud Rates and Character Formats

You may need to change the baud rate of the modem and the number of data bits, stop bits and parity of the data sent and received by the modem. The two most common combinations, 300 baud no parity and 1 stop bit and 110 baud with no parity and 2 stop bits are provided for by the firmware, but many more combinations are possible and some of them may be used by systems you want to communicate with.

Before you try to change these options, you should make sure that the Micromodem II has initialized itself. If you don't, then the first time you select the Micromodem II for input or output, it will initialize itself and change the options you have set back to the standard ones. The following line of BASIC will make sure that the initialization has been done:

```
100 PR#SLOT:PRINT
```

The baud rate is controlled by the least significant bit of the modem control byte. As described above, it is preferable to change the byte in memory and allow the firmware to take care of actually putting it in the hardware register. The following lines will change the baud rate without affecting any other modem functions:

To Select 300 Baud

```
500 POKE 1656+SLOT,PEEK(1656+SLOT)/2*2+1
```

To Select 110 Baud

```
500 POKE 1656+SLOT,PEEK(1656+SLOT)/2*2
```

The format of characters sent and received by the modem is controlled by bits 2, 3, and 4 of the ACIA control byte. Again it is preferable to change the byte in memory and then let the firmware take care of the actual hardware. This byte normally contains a 1 in the least significant bit plus the appropriate bits in bits 2, 3, and 4 to select the appropriate format. The following line of code can be used to set a character format:

```
700 POKE 2040+SLOT,FSW
```

The value of FSW can be selected from the following table:

Start Bit		Char. Length		Parity Bit		Stop Bits		Total Length		FSW Decimal	FSW Hex
1	+	7	+	EVEN	+	2	=	11	:	1	01
1	+	7	+	ODD	+	2	=	11	:	5	05
1	+	7	+	EVEN	+	1	=	10	:	9	09
1	+	7	+	ODD	+	1	=	10	:	13	0D
1	+	8	+	NONE	+	2	=	11	:	17	11
1	+	8	+	NONE	+	1	=	10	:	21	15
1	+	8	+	EVEN	+	1	=	11	:	25	19
1	+	8	+	ODD	+	1	=	11	:	29	1D

Waiting for the Nth Ring Before Answering

The following program fragment will wait for the Nth ring then answer the phone. It assumes that SLOT contains the slot number of the Micromodem II, and that the variable N contains the desired number of rings.

```
800 STT=-16251+16*SLOT
900 X=0
1000 IF PEEK(STT)>127 THEN GOTO 1000
1100 X=X+1
1200 IF X>=N THEN GOTO 1500
1300 IF PEEK(STT)<128 THEN GOTO 1300
1400 GOTO 1000
1500 IN#SLOT:INPUT I$
1600 IF PEEK(1656+SLOT)<127 THEN GOTO 900
```

Line 1000 waits for the phone to ring. Lines 1100 and 1200 count the rings. Line 1300 waits for the end of the ring before going back to 1000 to await the next ring. Line 1500 selects the Micromodem II for input and lets the firmware take care of answering the phone. Line 1600 then checks to see whether the firmware detected a valid carrier.

If you hang up on any program, the Micromodem II will hang up the phone as soon as it detects the loss of carrier. It will subsequently answer on the first ring and try to re-establish communications exactly where you left off. If you want your Apple II to go back to answering only after the Nth ring, you will have to make sure that your BASIC program understands that you are finished with it and hangs up on you so that it can then go back to waiting for the Nth ring.

Turning off the Carrier without Hanging Up

You may find it desirable to be able to turn off the modem carrier without breaking the telephone connection. For example you might write a game for two Apple II's. It would be nice if between rounds, you and your opponent could pick up your telephones and talk about the last round. With the modem transmitters running, all you would be able to hear would be the squealing of the two modems.

The following program fragments will allow your program to turn off the modem so that you can talk (pretty simple, actually), and then turn the transmitters back on and reestablish communication (this is a little more complicated). While the carriers are turned off, you must not attempt to execute any INPUT statements from the Micromodem II because if you do, the firmware will detect the loss of carrier and hang up the phone. So this program assumes that it is communicating with a similar program in another Apple II and that both programs know that it is time to turn off the carrier.

To Turn the Carrier Off

```
1000 POKE -16251+16*SLOT,136
```

To Turn the Carrier Back On and Reestablish Communications

```
1900 PR#0:IN#0:INPUT "PRESS RETURN WHEN DONE WITH PHONE",I$
2000 PR#SLOT:PRINT
2100 IF PEEK(-16250+16*SLOT) MOD 8 < 4 THEN GOTO 2400
2200 X=PEEK(-16249+16*SLOT)
2300 GOTO 2100
2400 PR#SLOT:PRINT "CONNECTION REESTABLISHED"
```

Line 1000 turns off the modem carrier by writing directly to the modem control port with a word containing the bits which keep the phone off hook and prevent the Micromodem II firmware from performing its initialization. Since the transmitter enable bit is off, the transmitter is turned off.

Line 1900 waits for you to press return when you are finished with the telephone. Line 2000 outputs a carriage return to the Micromodem II, which causes the firmware to copy its remembered status back into the modem control port. Since the last the firmware knew, the transmitter was turned on, this

turns the transmitter back on. Line 2100 reads the ACIA status port and checks the carrier detect bit to see if there is a carrier from the other end yet. Remember that we can't do an INPUT statement again until we have a valid carrier or the firmware will hang up the phone. Line 2200 unloads the ACIA data register. This obscure operation is necessary to satisfy the ACIA chip, and is pretty hard to explain but it is necessary. Line 2400 then sends a message to the other program so that it will know we have reestablished communication. In a real program you would probably send some other message to tell the other computer that it was time to get back to the game or whatever.

Entering Terminal Mode from a Program

It is often convenient to go back and forth between a BASIC program and terminal mode, especially when you are operating with two Apple II computers with Micromodem II's.

The terminal program in the Micromodem II firmware is entered from the input entry point if the TERM bit is set in the FLAG byte. So, for a program to activate the terminal mode, all it needs to do is to set that bit and then call for input from the Micromodem II. The following program fragment illustrates this technique.

```
1000 POKE 1912+SLOT,10
1100 INPUT I$
```

Location 1912+SLOT is the FLAG byte. The 10 is the sum of 8 + 2, where 8 is the binary weight of the TERM bit, and 2 is the binary weight of the KBDE bit. Whenever you go into terminal mode, you should be sure to set the KBDE bit or the Apple's keyboard will be inactivated except for ctrl-a sequences. Since we did not set the DISPO bit (weight 128), this program will start the terminal mode program in half-duplex (which is what you want to communicate with another Micromodem II in terminal mode). If you need to start it in full duplex mode all that is needed is to add the binary weight of the DISPO bit (128) to the constant which is POKE'd into the FLAG byte. Note that the binary weights of the bits are included in the tables at the back of the manual (p. 73).

If your program is communicating with another Micromodem II-equipped Apple II, it can also put the other computer into terminal mode by sending it a control-T. In order for this to work properly, the other Apple II must be waiting for input from its Micromodem II (be executing an INPUT or GET statement), and should not have the Micromodem II selected for output. The half/full-duplex status will depend on the setting of the DISPO bit in that computer.

Exiting Terminal Mode back to a Program

When you exit from terminal mode after entering it with this technique, you will return to the INPUT statement (line 1100 in the example). That input statement will be waiting for you to type a carriage return, just as always. As soon as you have satisfied it, your program will continue executing at the very

next line.

Your program can also take the other computer back out of terminal mode. All it needs to do is to send a control-R to the other Micromodem II. If the other computer got into terminal mode by setting the DISPO bit and executing an INPUT statement, then its program will be restarted if the control-R is followed by a RETURN. Like this:

```
1100 INPUT I$          <--- the same statement 1100 as above
1200 PR#SLOT
1300 PRINT CONRTOLR$
```

The string variable CONTROLR$ is assumed to contain a control-R. Since line 1300 did not end with a semicolon, BASIC will supply a RETURN at the end of the line.

INSPIRATIONAL PROGRAMS

The programs in this section, although useful in themselves are primarily intended to serve as inspiration for you to write better programs to serve your needs. They are designed to answer some of the questions we have answered during the first few months of Micromodem II production. Those questions usually begin "How do I ...".

This chapter has seven sample programs designed to solve some of the most frequently-encountered communications problems. The programs and the problems they solve are:

PROGRAM	PROBLEM
PICKUP	Picks up the phone in answer mode.
AUTO DIAL	Automatically dials numbers from a menu.
DUMBO	Dumb terminal written in BASIC.
TRANSFER	Transfers D.O.S. text files from Apple II to Apple II.
EXTRACT	Extracts a BASIC program from another system.
FILTER	Filters unwanted characters (such as ctrl-c) from input.
ALARM	A totally different use for the Micromodem II.

PICKUP

The Problem

If you are talking on the phone to a friend and decide that you would like to have your computers on the line, it would be nice to be able to have them pick up the phones and begin communicating. It is no problem to make a Micromodem II pick up the phone in originate mode. Simply go into terminal mode and dial a null phone number (i.e. just a RETURN). But for two modems to communicate with each other, one must be in originate mode and the other in answer mode. There is no simple way to make a Micromodem II pick up the phone in answer mode unless the phone actually rings.

What the Program Does

This program is designed to run in Applesoft II under D.O.S. With a D.O.S. system, all you need to do to pick up the phone is to type:

RUN PICKUP

Assuming, of course that you have previously stored this program on your disk under the name PICKUP. The program takes the phone off the hook, turns on the Micromodem II in answer mode, waits for a carrier, then puts you into terminal mode.

How the Program Works

Line 100 defines the slot the Micromodem II is in. Lines 300 and 400 select the Micromodem II for output and make sure that it is initialized by sending it a carriage return. Line 500 sets up the modem control word to be off hook in answer mode, and line 600 selects the standard character format. When line 650 transmits its carriage return, the Micromodem II firmware copies the data we POKE'd in 500 and 600 into the actual hardware registers, causing them to take effect.

Lines 700-900 wait for a carrier. This technique is described in ADVANCED PROGRAMMING (p. 38). Once a carrier is detected, lines 1000-1200 turn off Micromodem II output, enable Micromodem II input, and let the user know where things stand. The POKE at 1300 selects FLAG bits TERM and KBDE, which cause the Micromodem II firmware to go into half-duplex terminal mode when the INPUT statement is executed at line 1400. If you wanted to answer full-duplex, you could add 128, the binary weight of the DISPO bit to disable the local echo of characters.

PICKUP

```
0  REM PROGRAM TO PICK UP PHONE
1  REM IN ANSWER MODE
2  REM WRITTEN BY DON HYDE
3  REM COPYRIGHT 1979,
4  REM D.C. HAYES ASSOCIATES, INC.
90  ONERR  GOTO 9000
100 MSLOT = 3
200 D$ =  CHR$ (4)
300  PRINT D$;"PR#";MSLOT
400  PRINT
500  POKE 1656 + MSLOT,128+8+2+1
600  POKE 2040 + MSLOT,21
650  PRINT
700  IF  PEEK ( - 16250 + 16 * MSLOT)
     = 2 THEN 1000
800 X =  PEEK ( - 16249 + 16 * MSLOT)
900  GOTO 700
1000  PRINT D$;"PR#0"
1100  PRINT D$;"IN#";MSLOT
1200  PRINT "CONNECTION ESTABLISHED"
1300  POKE 1912 + MSLOT, 8 + 2
1400  INPUT I$
1500  END
9000  PRINT  CHR$ (26): PRINT D$;"PR#
      0"
9998  REM COPYRIGHT 1979,
9999  REM D.C. HAYES ASSOCIATES, INC.
```

ADD:

```
610 POKE -16250+16*MSLOT,21
```

AUTO DIAL

The Problem

Dialing up the systems we want to talk to would be much easier if the computer would remember their phone numbers for us and call them automatically.

What the Program Does

AUTO DIAL puts up a menu of telephone numbers and asks you to select one by its line number. If you select a valid line, it dials the phone. If it succesfully reaches another modem, it puts you into terminal mode, otherwise it asks if you would like to try another. If you ask for line 0, it assumes you want to dial manually and simply puts you into terminal mode. A negative line number exits the program.

The phone numbers on the menu included in this program are mostly Computerized Bulletin Board System's (CBBS's). They are mostly operated by computer hobbyists, and several of them use D.C. Hayes Associates, Inc. modems.

I would like to thank Ken Welk for this program and his generous permission to use it.

How the Program Works

This program runs under D.O.S. in Applesoft II. It could easily be modified to run without D.O.S., but it needs DATA statements, and would be much more difficult to write in integer BASIC.

After clearing the screen and initializing a few vairables, the program proceeds to READ the DATA statements into an array in lines 20-60. As they are READ, they are also PRINT'ed to create the menu. The first DATA statement tells how many lines of additional DATA statements to expect. The rest of the DATA statements each consist of two strings; a name, and its corresponding phone number.

Lines 70-100 get the line number and check it for validity. Then lines 110-115 dial the phone. Line 116 checks whether the call was successful. Lines 120-140 put you into terminal mode.

When you exit from terminal mode, you will return to the INPUT statement in line 140. If you type a RETURN, you will satisfy that statement and the program will continue execution at line 150 where it will ask whether you want to place another call.

AUTO DIAL

```
5  TEXT : HOME
10 Q$ = CHR$(17):D$= CHR$(4)
15  PRINT D$;"NOMON I,O,C"
20  READ NC
30  DIM PN$(NC,2)
35  PRINT "CBBS'S & OTHER SYSTEMS": PR
INT "------ - ----- -------": PRINT
40  FOR I = 1 TO NC
50  READ PN$(I,1),PN$(I,2)
55  PRINT I; TAB( 6);PN$(I,1); TAB( 24
);PN$(I,2)
60  NEXT
70  INPUT "YOUR CHOICE?";CH
80  IF CH < 0 THEN  END
90  IF CH > NC THEN 70
100  IF CH = 0 THEN 120
110  PRINT D$;"PR#3"
115  PRINT Q$;" ";PN$(CH,2)
116  IF  PEEK (1659) < 128 AND CH < >
     10 THEN  PRINT D$;"PR#0":
PRINT "NO ANSWER OR BUSY! ":
     POKE -16368,0: GOTO 150
120  PRINT D$;"PR#0"
130  PRINT D$;"IN#3"
140  POKE 1915,142: INPUT I$
150  PRINT D$;"IN#0": INPUT "ANOTHER C
ALL?";A$: IF  LEFT$ (A$,1) = "Y"
     THEN RUN
160  END
200  DATA 13
215  DATA   ATLANTA,1-(404)-394-4220
220  DATA   BOSTON,1-(617)-963-8310
230  DATA   CHICAGO,1-(312)-528-7141
240  DATA   S.J.C.A.,1-(609)-665-8881
250  DATA   DALLAS,1-(214)-641-8759
260  DATA   MAYNARD,1-(617)-897-0190
261  DATA   PASADENA,1-(213)-795-3788
265  DATA   SAN FERNANDO,1-(213)-340-
     0135
266  DATA   SANTA CLARA,1-(408)-246-
     2805
267  DATA   WASHINGTON,1-(703)-281-
     2125
270  DATA SAN DIEGO CBBS,1-(714)-565-0
     961
280  DATA SAN DIEGO ABBS,1-(714)-582-9
     557
285 DATA SAN DIEGO C.S.,1-(714)-697-2
     176
900  REM  THANKS TO KEN WELK
910  REM  IN SAN DIEGO FOR
920  REM  THIS USEFUL PROGRAM.
```

DUMBO

The Problem

Although the dumb terminal program included in the Micromodem II's ROM is pretty general and was designed to accomodate most of the odd requirements you might run into when communicating with various systems, it can't possibly do everything for everybody. If it doesn't do just what you need, all is not lost for it is quite possibly (though not particularly easy) to write one in BASIC.

Several customers have expressed an interest in writing some sort of intelligent terminal program that would use the power available from the Apple II, but they didn't know quite where to start. This should help, because I have already worked out some of the harder parts.

What the Program Does

After asking a few questions to determine what options you want, this program dials the phone number you have supplied then enters a loop in which it directly interrogates the keyboard and the modem input port and transfers bytes between them, the screen, and the modem output. It checks for error conditions on the modem, and reports them to the user.

It does not check any of the characters coming from either the keyboard or the modem with one exception. It detects ctrl-g (BELL) from the modem, and calls a special short beep routine. It seems that the beep the Apple II firmware generates is 100 msec. long, which is 3 character times at 300 baud. This means that each time the remote computer sends a BELL, the program will detect several errors when it misses the two characters following the beep.

This program easily keeps up at 300 baud, and has a fair amount of time left over, which means that you could add logic to test for various control codes and still have a program which will keep up with the stream of data arriving from the modem.

How the Program Works

The left column is pretty much all initialization, inclucing dialing the phone, and the right column is the terminal loop, including the modem error handler.

Most of the initialization is pretty straightforward except for a couple of things. Lines 100 and 135 set up a simulated CHR$ function, which makes it easier to print the data we have obtained via PEEK's as characters rather than as numbers. The technique used here was described in more detail in CONTACT 2 (p. 7).

Line 151 installs a machine-language program that makes short beeps. It simply loads a smaller constant into the Y register then jumps into the beep-making routine in the Apple II's monitor ROM. The machine code is:

```
300: LDY     #$23
302: JMP     $FBE4
```

Lines 155-185 are pretty straightforward. Then we go into the terminal loop from lines 200-9500. 200-240 are the status-checking loop. Lines 250-275 handle characters read from the keyboard, data is displayed on the screen on lines 1000-1020, input from the modem is handled by the 5000's and the 9000's handle errors.

If you examine the STATUS register, you will see that during normal operation, only the two least significant bits have any reason to be on. Therefor if line 210 finds a value greater than 3, it knows that there is some error condition to take care of. The expression X MOD 2 in line 220 effectively tests X for being odd. If it is odd that means that the least significant bit is a one. That in turn means that there is a character ready in the receiver data register, so the program goes to 5000 to handle it.

Line 230 reads the keyboard port, line 240 tests whether there is a character there, and line 250 releases it if there is one. Lines 260 and 270 transmit the character via the Micromodem II.

Line 1000 is the simulated CHR$, and line 1010 displays the character on the Apple II's screen.

A character is read from the Micromodem II in line 5000. Line 5005 makes sure that the most significant bit is set. This makes sure that we will recognize the characters correctly no matter what parity option has been selected. Line 5010 looks for BELL characters. Most characters are displayed by the code at 1000, but BELL's are handled specially by calling our custom short beep routine located at $300 = 768 dec.

The first thing the error routine looks for at line 9000 is loss of carrier. We single this error out by a quick test based on our knowledge that its binary weight is 4. We know that the program could only get here if X>3 (line 210). If the carrier has been lost, the X value will be between 4 and 7 (assuming that no other error bits are set). It is possible that other error bits may get set when carrier is lost, but they will go away when the error is cleared, and the carrier will still be lost so we'll catch it next time.

If the error is not a loss of carrier, we read the data register, which clears the error condition, and give the user an error message before we go back to the terminal loop.

When we detect a loss of carrier, we select the Micromodem II for output and send it a string of characters (at line 9100) which contains a couple of spaces followed by a control-z. The control-z hangs up the phone. The spaces are a good idea when you hang up because they insure that the last useful characters have been transmitted before the phone is hung up. The ACIA chip buffers a couple of characters, and whatever is receiving the data at the other end of the line probably buffers a character or two also, so there could still be data on its way for several character times after you sent your last

character.

DUMBO

```
  0 REM DUMB TERMINAL PROGRAM
  1 REM IN APPLE II INTEGER BASIC
  2 REM WRITTEN BY DON HYDE
  3 REM COPYRIGHT 1979,
  4 REM D.C. HAYES ASSOCIATES, INC.
  5 REM
100 A$="X"
105 X=0
110 SLOT=3
115 CR1=-16250+16*SLOT
120 DTA=CR1+1
125 CHAR=1912
130 SEND=-16382+256*SLOT
135 AD=2053
140 D$=""
145 KBD=-16384
150 KBRL=KBD+16
151 POKE 768,160: POKE 769,35: POKE
770,76: POKE 771,228: POKE 772,251
153 DIM PH$(25)
155 PRINT "DUMB TERMINAL PROGRAM"
160 INPUT "HALF OR FULL DUPLEX?  TYP
E H OR F. ",A$
165 F=0: IF A$="F" THEN F=1
170 IF F=0 AND A$#"H" THEN 160
175 INPUT "PHONE NUMBER?",PH$
180 PRINT D$;"PR#";SLOT: PRINT "";P
H$
185 PRINT D$;"PR#0"
200 X= PEEK (CR1)
210 IF X>3 THEN 9000
220 IF X MOD 2 THEN 5000
230 X= PEEK (KBD)
240 IF X<128 THEN 200
250 POKE KBRL,0
260 POKE CHAR,X
270 CALL SEND
275 IF F THEN 200
1000 POKE AD,X
1010 PRINT A$;
1020 GOTO 200
5000 X= PEEK (DTA)
5005 IF X<128 THEN X=X+128
5010 IF X<>135 THEN GOTO 1000
5020 CALL 768
5030 GOTO 200
9000 IF X<8 THEN 9095
9010 X= PEEK (DTA)
9015 PRINT : PRINT "ERR, DTA= ";X
9020 GOTO 200
9095 PRINT D$;"PR#";SLOT
9100 PRINT "     "
9200 PRINT D$;"PR#0"
9300 PRINT "LOST CARRIER!"
9500 GOTO 155
9998 REM COPYRIGHT 1979,
9999 REM D.C. HAYES ASSOCIATES, INC.
```

TRANSFER

The Problem

To be able to transfer text files from one Apple II to another.

What the Program Does

This Applesoft II program is designed to communicate with another copy of itself which is loaded into another Apple II computer. When it starts, it puts you into terminal mode so that you can communicate with the operator of the other Apple II and establish who will send what to whom.

Once you have agreed on what to send, either one of you may type ctrl-a ctrl-x to exit terminal mode, followed by a RETURN. Both computers will then

ask for a file name, and will open the appropriate disk file. Then they will ask whether you wish to SEND or to RCVE a file. As soon as both operators have answered both questions, the programs will start transferring the file one line at a time, with the data being displayed on both screens as it is being sent. The two programs communicate back and forth to make sure that they stay in sync with each other so that no data will be lost.

When the last line has been sent, the sending computer sends a ctrl-c to the receiving computer, which tells it that all the data has been sent, then both machines close their files and go back into terminal mode.

How the Program Works

After a little initialization, the program goes from line 300 to line 3500, which puts the Micromodem II into terminal mode. After the exit from terminal mode and the entry of a RETURN, the program continues at 3650, which transmits a ctrl-r followed by a RETURN, which takes the other computer out of terminal mode and satisfies its statement 3600, thus restarting it.

By now you might have already noticed the use of the handy subroutines at lines 400-700 which save a lot of writing in doing the IN# and PR# statements.

The actual work starts at 900. Asking the questions and opening the file is easy. Then we go into either the sending loop (lines 1500-2500) or the receiving loop (lines 4000-4800).

The first thing the sending loop does is to execute an ONERR statement so that it will be able to trap the OUT OF DATA error when it gets to the end of the file. It then enters its loop at 2300. There, it de-selects the Micromodem II for output and selects it for input. Line 2400 then waits until it gets an ACK character, discarding any other data it may receive. It is waiting for the receiving loop in the other computer to tell it that it is ready for some data. The PRINT in line 2500 is needed to make sure that D.O.S. will respond correctly, then we go back to 1800.

Next we turn off the Micromodem II input, and turn on the output. Line 1850 tells D.O.S. that we want to read data from the file, and lines 1900-2100 read it into I$. This method using GET statements makes it possible to read data which contains commas and D.O.S. commands. Line 2200 sends the line to the other computer. The D$ is needed to cancel the D.O.S. READ. This gets us back to 2300 where we started.

When we get to the end of the file, the ONERR is executed and control goes to line 3000 where we first check to make sure that we reached end of file and not some other error condition. If it was end of file, then we send our ctrl-c to let the other guy know we're done, tell the user we're done, and return to terminal mode.

The receiving loop begins at 4000 by issuing a PRINT to make sure that D.O.S. will hear us properly. Then we ask D.O.S. to display the data we will be writing to the disk. At line 4050, we call FRE to make sure that our string

space is all in order and that we won't suddenly lose precious time when Applesoft II has to go collect unused string space. If that happened, we would almost surely lose some characters.

At line 4100, we select local display for output, and Micromodem II for input. Then we set I$ to the null string preparatory to filling it with data we receive from the Micromodem II. We are now ready to start receiving data, so we transmit an ACK character at line 4250. Sending it this way helps to keep from getting D.O.S. confused. Then we start GET'ting the characters in line 4300. We check each character for end of line (CR$) or end of file (EF$) and collect the rest of them into I$. When we see a CR$, we write the line to disk in lines 4600 and 4700, then go back to 4050 to get ready for another line.

When we recognize the ctrl-c (EF$) that marks the end of the file, we go to 5000 to close the file, then we go back into terminal mode.

TRANSFER

```
0  REM  TEXT FILE TRANSFER PROGRAM
1  REM  WRITTEN BY DON HYDE
2  REM  COPYRIGHT 1979 D.C. HAYES
         ASSOCIATES, INC.
100 MSLOT = 3
200 D$ =  CHR$ (4):AK$ =  CHR$ (6):
    CR$ =  CHR$ (13)
225 EF$ =  CHR$ (3):RE$ =  CHR$ (18)
250  PRINT D$;"NOMONI,O,C"
300  GOSUB 500: GOTO 3500
400  PRINT D$;"IN#0": RETURN
500  PRINT D$;"PR#0": RETURN
600  PRINT D$;"IN#";MSLOT: RETURN
700  PRINT D$;"PR#";MSLOT: RETURN
900  GOSUB 400: GOSUB 500
1000  INPUT "FILE NAME? ";F$
1100  PRINT D$;"OPEN ";F$
1200  INPUT "SEND OR RCVE? ";I$
1300  IF I$ = "RCVE" THEN 4000
1400  IF I$ < > "SEND" THEN 1200
1500  ONERR  GOTO 3000
1550  POKE 1912 + MSLOT,2
1600  GOTO 2300
1800  GOSUB 700: GOSUB 400
1850  PRINT D$;"READ ";F$
1900 I$ = ""
2000  GET A$: IF A$ = CR$ THEN 2200
2100 I$ = I$ + A$: GOTO 2000
2200  PRINT D$;I$
2300  GOSUB 500: GOSUB 600
2400  GET A$: IF A$ < > AK$ THEN 2400
2500  PRINT : GOTO 1800
3000 X =  PEEK (222): POKE 216,0
3100  GOSUB 400: GOSUB 500
3200  IF X < > 5 THEN 9000
3300  GOSUB 700: PRINT EF$;EF$: GOSUB
      500
3400  PRINT : PRINT "FILE SENT"
3500  GOSUB 600: POKE 1912 + MSLOT,8 +2

3600  INPUT F$
3650  GOSUB 700: PRINT RE$
3700  GOTO 900
4000  PRINT
4025  PRINT D$;"MON O"
4050 X =  FRE (0)
4100  GOSUB 500: GOSUB 600
4200 I$ = ""
4250  POKE 1912,134: CALL  - 16382 +
      256 * MSLOT
4300  GET A$: IF A$ = CR$ THEN 4600
4400  IF A$ = EF$ THEN 5000
4500 I$ = I$ + A$: GOTO 4300
4600  PRINT : PRINT D$;"WRITE ";F$
4700  PRINT I$
4800  GOTO 4050
5000  PRINT : GOSUB 400
5100  PRINT D$;"CLOSE ";F$
5200  PRINT "FILE RECEIVED"
5300  GOTO 3500
9999  REM  COPYRIGHT 1979 D.C. HAYES
       ASSOCIATES, INC.
```

BASICEX

The Problem

If you use a time-sharing system and have created some useful BASIC programs, you might well want to move them over to your Apple II computer and modify them so that you can run them there. This can be fairly challenging to do since time-sharing systems are usually not set up to be able to transmit BASIC programs to another computer, and the BASIC interpreters in the Apple II are not designed to be able to accept programs from another computer.

What the Program Does

This program is designed to extract a BASIC program from another dialup computer and save it in a D.O.S. text file. Once you have a BASIC program in a text file, you can manipulate it with a BASIC program of your own (to make simple changes such as changing semicolons to colons), and you can present it to Applesoft II by submitting it as an EXEC file.

As you might have noticed, Applesoft II's input editor will allow you to enter and edit programs which cannot possibly be run in Applesoft II. This could be convenient in this application because you can then use all your normal program editing facilities to work on the foreign program while you are converting it to Applesoft II.

The BASIC program is extracted by issuing a series of LIST commands. It is assumed that you have some knowledge of the line numbering of the program you are attempting to extract, preferably the exact knowledge that would come from having used the RENUMBER command which many BASIC's have.

The first thing the program does is to ask you for the low line number, the high line number, and the line increment. It uses the information you supply to issue a series of LIST commands that will each ask for about 10 lines. It has a buffer that will hold 30 lines, so if you are off by a little, it won't blow up, but it does help to have an accurate guess.

One important assumption this program makes is that the BASIC interpreter it is conversing with puts out some recognizable sequence of characters when it is finished doing a LIST command so that the program can know when the BASIC interpreter is done so that it can go put those lines away on the disk. This program was tested dialing into a system using Microsoft BASIC, which puts out a line that says "ok" whenever it has finished with any command.

Most likely, whatever system you hope to use this program with will be different in this respect, and you will have to make some changes to this part of the program. I have also written and tested a version which recognizes the] prompt which Applesoft II issues. These two cases; a line containing some recognizable message, and a prompt character will probably cover most situations.

The changes for Applesoft II extraction are:

```
1400 OK$="]"
2415 J=0
2480 IF J<3 AND A$=OK$ THEN 2850
2485 J=J+1
2600 delete
```

How the Program Works

There's nothing very complicated up to line 2050. This line sets the FLAG byte to all zeroes, which enables the Micromodem II's local display of output and disables the local keyboard. Line 2100 instructs Applesoft II to perform storage reclamation which will insure that this time-consuming process does not happen unexpectedly when we are trying to keep up with the data coming at us from the other computer.

The POKE at 2250 causes the LIST command which line 2300 transmits to the other computer to appear on our screen in reverse video. The POKE at 2350 restores normal video. Line 2400 turns the Micromodem II around, i.e. instead of outputting to the Micromodem II, we input from it.

As you may recall, it takes a finite amount of time for the characters we send to reach the other end of the line, mostly due to the delays inherent in converting it into a serial bit stream. By the time we reach line 2402, there are probably one or two characters from line 2300 which have still not reached the other computer. Since most computers echo each character as it is received, when we reach this point, the data which we see arriving from the Micromodem II is just our own data being echoed back to us. Line 2402 simply waits until it sees the RETURN, which was the last character that we sent. If the system you are communicating with operates half-duplex, that means that it does not echo characters so you will want to delete line 2402.

Lines 2410-2510 GET (line 2425) characters as they arrive from the Micromodem II, display them, and append them to the current line (I$(L)). This complication is necessary because if we tried to use an INPUT statement, Applesoft II would get upset about the commas that are bound to occur in any BASIC program. At line 2600 we check for the ok which tells us we've got all we're going to in this batch of lines.

Once we have a batch of lines, lines 2900-3400 write them onto the disk. Then at line 3300 we compute the next line number to ask for, check to see if we've gotten them all at line 3600, and if necessary go back to 2100 for another batch. If we've got them all, we close the file, issue a message, and put the Micromodem II into terminal mode.

Lines 9000-9300 are a convenient ONERR routine which switches our input and output back to the keyboard and console and prints an error message if anything goes wrong while we are communicating. This will make it easier to fix it,

since we know we don't have to worry which computer we are typing to.

HINT: If you are debugging a modified version of the program, it could be helpful to enable the local keyboard (line 2050) and add a test for control-c, so that you can stop the program if it doesn't seem to be behaving properly.

BASICEX

```
0  REM BASIC PROGRAM EXTRACTOR
1  REM WRITTEN BY DON HYDE
2  REM COPYRIGHT 1979,
3  REM D.C. HAYES ASSOCIATES, INC.
8  ONERR  GOTO 9000
9  CLEAR
10 CR$ =  CHR$ (13)
20  GOTO 1000
100  PRINT D$;"IN#0": RETURN
200  PRINT D$;"PR#0": RETURN
300  PRINT D$;"IN#";MSLOT: RETURN
400  PRINT D$;"PR#";MSLOT: RETURN
1000 ML = 10
1100  DIM I$(ML * 3)
1200 D$ =  CHR$ (4)
1250  PRINT D$;"NOMONI,O,C"
1300 MSLOT = 3
1400 OK$ = "OK"
1500  GOSUB 100: GOSUB 200
1600  INPUT "LOW LINE NUMBER? ";LL
1700  INPUT "HIGH LINE NUMBER? ";HL
1800  INPUT "LINE INCREMENT? ";LI
1900  INPUT "FILE NAME? ";F$
2000  PRINT D$;"OPEN ";F$
2050  POKE 1912 + MSLOT,0
2100 X =  FRE (0)
2200 L = 0
2250  POKE 50,63
2300  GOSUB 400: PRINT "LIST ";LL;"-";
 INT (LL + (ML - 1) * LI)
2350  POKE 50,255
2400  GOSUB 200: GOSUB 300
2402  GET A$: IF A$ <  > CR$ THEN 2402
2410 I$(L) = ""
2425  GET A$
2450  IF A$ = CR$ THEN 2575
2475  PRINT A$;
2500 I$(L) = I$(L) + A$
2510  GOTO 2425
2575  PRINT
2600  IF I$(L) = OK$ THEN 2900
2700 L = L + 1
2800  GOTO 2410
2900  GOSUB 100
3000  PRINT D$;"WRITE ";F$
3100  FOR I = 0 TO L - 1
3200  PRINT I$(I)
3300  NEXT I
3400  PRINT D$
3500 LL = LL + (ML - 1) * LI + 1
3600  IF LL < HL THEN 2100
3700  PRINT D$;"CLOSE ";F$
3800  PRINT "FILE SAVED ON DISK."
3900  GOSUB 300: POKE 1912 + MSLOT,128
      + 8 + 2
4000  INPUT F$
4100  GOTO 1500
9000  GOSUB 100: GOSUB 200
9100  PRINT "ERROR NUMBER "; PEEK (222)
9200  PRINT "AT LINE "; PEEK (218) +
      PEEK (219) * 256
9300  END
9998  REM COPYRIGHT 1979,
9999  REM D.C. HAYES ASSOCIATES, INC.
```

FILTER

The Problem

Sometimes we have an application program which we would like to be able to put on-line so that other people can use it. This sometimes presents a problem

because some people are more interested in crashing the system than they are in using it. If you put a BASIC program on-line, it is pretty easy to crash it if you type a control-C and wipe out the program. If you are using Applesoft II, you can issue an ONERR statement, which will trap out control-C's and any other error the caller might provoke. But this only works for Applesoft II, and requires that each program you put on-line be modified.

What the Program Does

This one is actually two programs. The filter is pair of small machine-language routines. One is a custom IN# routine which modifies D.O.S. pointers so that the filter routine is inserted between the Micromodem II and D.O.S.'s input.

The other program is a BASIC program which installs and initializes the machine-language routines. Once they are installed, to select the Micromodem II for input, you should CALL 768 instead of doing the usual IN# statement. Note that it is not necessary to put the CALL statement into a PRINT statement as you would do with the IN# statement.

How the Programs Work

The BASIC program is pretty simple, consisting mostly of REM statements. It asks D.O.S. to load the machine-language program at line 400, at 550 it modifies the first instruction of the machine-language program to correspond to the correct slot number for the Micromodem II, and at 600 it calls the custom IN# routine. If you put your Micromodem II in another slot, all you need to change is line 100.

The machine-language routines reside in an area of memory reserved for just such small routines as this one. This area is not used by either specie of BASIC or by D.O.S. except during a boot, so once it is installed, it will not go away unless you or one of your programs write something else there.

The custom IN# routine occupies locations $300-$327. It first loads a constant ($C3 for slot 3) which reflects the Micromodem II's current slot number. Next it uses this constant to modify the JSR instruction at location $32A. It then makes sure that the TRAN (code transparency) bit is set in the FLAG word. This disables the potentially dangerous control-Y function of the Micromodem II firmware. It then computes an address based on a pointer which D.O.S. maintains at locations $3E7 and $3E8 and stores the result in locations $2A and $2B. It then indirectly stores the address of the filter routine at this computed address. This effectively installs the filter routine between D.O.S. and the Micromodem II firmware.

If you are not using D.O.S., the section from $30D-$325 could be replaced by a much simpler one that simply stores the address of the filter routine in locations $38 and $39 (KSWL and KSWH).

The filter routine itself occupies locations $328-$33C. It first saves the

Y register on the stack, then calls the Micromodem II input routine. When the Micromodem II firmware returns, there is a character in the A register. It then loads a loop count (4) into the Y register and loops through a table starting at location $33D, comparing each table entry to the contents of the A register. If a match is found, that means that the character in the A register is one of the discard characters, so it branches back to the JSR at #32A to get a new one from the Micromodem II. If the character does not match any of those in the discard table, then it restores the Y register, destroying the A register in the process. Fortunately, the Micromodem II firmware left a copy of the character in location $778, so we are able to restore that before we return to D.O.S. with our safe character in the A register.

If you wish to modify the discard list or add to it, you may do so by simply entering the hex values for the characters you wish to filter out into the table with the monitor. If you add more characters to the table, be sure to change the loop counter (location #32E). The loop counter contains the count of the characters in the discard table. If you modify the table, you will probably want to BSAVE it as described in line 26 of the BASIC program.

FILTER

```
0 REM INSTALL FILTER
1 REM WRITTEN BY DON HYDE
2 REM COPYRIGHT 1979,
3 REM D.C. HAYES ASSOCIATES, INC.
10 REM
11 REM THIS PROGRAM INSTALLS A
12 REM FILTER ROUTINE WHICH FILTERS
13 REM OUT SOME CHARACTERS FROM
14 REM MODEM INPUT.
15 REM
16 REM THE MACHINE-LANGUAGE FILTER
17 REM ROUTINE OCCUPIES 50 HEX
18 REM BYTES STARTING AT 300 HEX.
19 REM
20 REM ITS TABLE OF DISCARD CHARACT
     ERS
21 REM BEGINS AT HEX 33D=829 DEC.
22 REM THE LENGTH OF THE TABLE
23 REM IS AT HEX 32E=814 DEC.
24 REM IF YOU MODIFY IT, SAVE IT
25 REM BY TYPING:
26 REM BSAVE FILTER,A$300,L$50
27 REM
28 REM ONCE IT IS INSTALLED, INSTEAD
29 REM OF IN#MSLOT, DO CALL 768
30 REM
100 MSLOT=3: REM  MODEM SLOT NUMBER
200 D$="": REM CTRL-D!
300 PRINT D$;"PR#";MSLOT
400 PRINT D$;"BLOAD FILTER"
500 PRINT
550 POKE 769,MSLOT+192
600 CALL 768
700 END
9998 REM COPYRIGHT 1979,
9999 REM D.C. HAYES ASSOCIATES, INC.
```

THIS IS THE FILTER PROGRAM ITSELF

```
0300-   A0 C3      LDY   #$C3
0302-   8C 2C 03   STY   $032C
0305-   B9 B8 06   LDA   $06B8,Y
0308-   09 04      ORA   #$04
030A-   99 B8 06   STA   $06B8,Y
030D-   38         SEC
030E-   AD E7 03   LDA   $03E7
0311-   E9 6B      SBC   #$6B
0313-   85 2A      STA   $2A
0315-   AD E8 03   LDA   $03E8
0318-   E9 00      SBC   #$00
031A-   85 2B      STA   $2B
031C-   A0 00      LDY   #$00
031E-   A9 28      LDA   #$28
0320-   91 2A      STA   ($2A),Y
0322-   A9 03      LDA   #$03
0324-   C8         INY
0325-   91 2A      STA   ($2A),Y
0327-   60         RTS
0328-   98         TYA
0329-   48         PHA
032A-   20 07 C3   JSR   $C307
032D-   A0 04      LDY   #$04
032F-   D9 3C 03   CMP   $033C,Y
0332-   F0 F6      BEQ   $032A
0334-   88         DEY
0335-   D0 F8      BNE   $032F
0337-   68         PLA
0338-   A8         TAY
0339-   AD 78 07   LDA   $0778
033C-   60         RTS
033D-   80
033E-   FF
033F-   83
0340-   84
0341-   00
```

ALARM

The Problem

Isn't there anything useful you can do with one of these things?

What the Pogram Does

I call this program a remote alarm clock. It might better be described as a computerized wake-up call. It uses a Micromodem II and a Mountain Hardware Apple Clock to make a phone call at a specified time and generate a distinctive sound when the phone is answered. This could be a useful program, though I am somewhat hesitant to publish it because I fear that it has great nuisance potential.

How the Program Works

After some initialization, the program reads the clock at line 1000 and displays the current time at line 1200. It then gets (and checks) a time and a phone number. At line 2000, it builds a display which tells what it is doing. Lines 2200-2700 are a loop in which the program reads the clock, updates the time on the display, and checks to see if the time has arrived.

When the time arrives, the program places the phone call in line 2800 (note the use of the line feed character (LF$) in the phone number). The loop in lines 2900-3600 generates a distinctive sound which is sort of like a European

siren by switching the Micromodem II back and forth between originate and answer modes. Line 3000 turns the Micromodem II's transmitter off but keeps the phone off the hook by POKE'ing a byte containing just the off-hook bit (OH). Line 3100 then turns it back on in the originate mode by setting the OH, MODE, and TXE bits. After a short delay, line 3300 turns the transmitter off again and line 3400 turns it back on in the answer mode. After another short delay, the process repeats, and after 30 seconds or so, the program drops through to line 3700 and hangs up the phone.

It is necessary to turn the modem transmitter off before you change the mode or it will not operate properly. There are actually four tones the Micromodem II can generate, a MARK tone and a SPACE tone in each of the two bands. Normally, when the Micromodem II is not transmitting any data, it continually generates the MARK (higher frequency) tone of the band which is selected. To switch to the lower tone, you must POKE a 96 dec. into the ACIA control register (CR1).

ALARM

```
0 REM TELEPHONE ALARM CLOCK PROGRAM
1 REM IN APPLE II INTEGER BASIC
2 REM USING MICROMODEM II AND
3 REM APPLE CLOCK.
4 REM WRITTEN BY DON HYDE
5 REM COPYRIGHT 1979,
6 REM D.C. HAYES ASSOCIATES, INC.
100 REM
200 DIM I$(25),PH$(10),HA$(2)
300 DIM MA$(2),HT$(2),MT$(2)
400 TSLOT=4: REM  APPLE CLOCK
500 MSLOT=3: REM  MICROMODEM II
600 CR2=-16251+16*MSLOT: REM  MODEM
    CONTROL
700 DIAL$="":BYE$="":LF$=""
800 CALL -936
900 PRINT "REMOTE ALARM CLOCK":
    PRINT
1000 PR#TSLOT: IN#TSLOT: INPUT " ",I$
1100 IN#0: PR#0
1200 PRINT "TIME NOW IS ";I$
1300 PRINT : INPUT "TIME FOR ALARM --
HH:MM ",I$
1400 IF LEN(I$)#5 THEN 1200
1500 IF I$(3,3)#":" THEN 1200
1600 INPUT "PHONE NUMBER TO RING ",PH$
1700 HA$=I$(1,2)
1800 MA$=I$(4,5)
1900 CALL -936
2000 VTAB 10: PRINT PH$;" WILL RING A
     T ";HA$;":";MA$
2100 VTAB 15: TAB 1: PRINT "TIME NOW
     IS"
2200 PR#TSLOT: IN#TSLOT: INPUT " ",I$
2300 HT$=I$(7,8)
2400 MT$=I$(10,11)
2500 PR#0: VTAB 15: TAB 13: PRINT HT$
     ;":";MT$: VTAB 23: TAB 1
2600 IF HA$=HT$ AND MA$=MT$ THEN 2800
2700 GOTO 2200
2800 PR#MSLOT: PRINT DIAL$;PH$;LF$;
2900 FOR I=1 TO 50
3000 POKE CR2,128
3100 POKE CR2,134
3200 FOR J=1 TO 100: NEXT J
3300 POKE CR2,128
3400 POKE CR2,130
3500 FOR J=1 TO 100: NEXT J
3600 NEXT I
3700 PRINT BYE$
3800 IN#0: PR#0
3900 GOTO 800
9000 REM COPYRIGHT 1979,
9001 REM D.C. HAYES ASSOCIATES, INC.
```

BACKGROUND INFORMATION

The information in this chapter is not essential to using the Micromodem II. But I believe that you will find it to be helpful and possibly even interesting. Data communications is a fairly complex topic, combining as it does aspects of several related technologies, each of which is a complex and interesting subject in itself.

Data communications is a very confusing subject for most people, including data processing professionals. The subject in inherently complex, and the terminologies are often misleading. This is partly because of the mix of data-processing, electronic, and telephone technologies. Frequently the same words mean different things in the different fields, and when words from the different fields are combined to describe concepts which combine the different fields, they often carry a built-in confusion factor.

I have chosen several of the most important and most confusing topics and have attempted to describe them in plain English. I hope that I have succeeded at least to some extent.

Compatibility With the Bell System 103 Modem

The Micromodem II is designed to be completely compatible with the communiation frequencies and modulation techniques of the Bell System (Western Electric) model 103 low-speed modem. The Bell System 103 (and its various equivalents) is by far the most widely-used modem in North America. It is used by virtually all time-sharing systems and dial-up data access systems as their standard mode of access. This popularity is due partly to the relative simplicity of the 103's FSK modulation technique and the reasonable cost of the circuitry required to implement it and partly due to the fact that there are so many other 103-compatible modems already installed to talk to.

What IS a modem anyway?

Early in the history of computing (back in the dim distant days of the 1950's),when computers will still huge like dinosaurs and people were just beginning to discover their data processing power, it occurred to someone that you could do a lot of neat things if you had two computers with a wire between them, or a computer and a typewriter if they had a wire connecting them. When they started thinking about wires that ran across the country, they soon began to think about how nice it would be if they could use all the phone wires that were already there.

Unfortunately, as we all know, computers talk digital and telephones are designed to carry the human voice. And the Bell System has lots of smart people who've been working for years and years building black-magic widgets of all kinds that mash and tear up those voice signals in all sorts of inconceivable ways to get them into long-distance wires in the most efficient possible way.

They spent decades researching the human voice so that they knew exactly how much they could mash and distort it and still have it come out recognizable at the other end of the line. If you could send digital data on the phone, who knows what all those widgets might do to it before it got to the other end of the line.

The phone company was worried that digital signals might hurt their various widgets, or might interfere with normal voice signals, so they weren't very encouraging at first.

But the problem did get some study, and some experiments were carried out (mostly by the Bell System). It seems that one thing all the widgets are careful not to disturb is the frequencies of the tones making up a voice. They may distort the amplitude or the phase, but they are careful not to distort frequencies. So a device was built which encoded digital data consisting of ones and zeroes into a signal containing different frequencies for ones and zeroes. This was called a modulator. That was pretty easy. The hard part was building something to un-do the modulating, and turn the tones back into digital ones and zeroes. Suffice it to say that they did figure out a way. They called the un-doer a demodulator.

It turned out that the modulator and the demodulator worked pretty good (better than most people expected). So the phone people put one of each into a box, and started to sell it as data transmission service. Like most engineering types, they weren't at their best with words, so they just took pieces of the words for the parts and stuck them together to make a word for the box. MOdulator + DEModulator = MODEM, thus the modem was born.

Well, a lot of things have happened since those distant days when dinosaurs first spoke to each other over the phone. A lot of research was done on modems. They got faster. And they got bigger and more expensive. In fact, the speed and price have tracked very closely, and there is a common rule of thumb that modems cost about a dollar a baud. But that didn't matter much because the computers they worked with were even bigger and more expensive.

Things happened even faster when the Supreme Court decided to allow competition in the modem business.

At first the phone company insisted that they had a legal monopoly in the modem business because modems were part of the telephone system and they had a legal monopoly to run the phones. Nobody argued. After all, it was pretty obvious that a modem had to be wired up to the phone line, and that certainly made it part of the phones didn't it? But what about the computer wired to the modem? Certainly it wasn't a telephone and the phone company couldn't claim a monopoly on making computers, too. Then along came the acoustic coupler, a modem that wasn't wired up to the phone line. It talked into the phone just like a human, making noises into the mouthpiece, and listening to the noises coming from the earpiece.

The phone company insisted that an acoustic coupler was also a violation of their protected monopoly. The Carterfone company, which built an acoustically-coupled device for use with two-way mobile radios, went to court

after the phone company put them out of business by telling their customers that the acoustic coupler was illegal and threatening to disconnect their telephones if they didn't stop using it.

The court eventually decided in favor of Carterfone (awarding them considerable damages, which helped them to get back in business), and ruled that acoustic couplers were legal. More recent decisions have broadened the rules for interconnect, giving the FCC the power to license devices for use on the telephone network in much the same way that they have long licensed radio transmitters, and limiting the phone company's protected monopoly to running the network which connects the phones together.

Today the dinosaurs, though still very much alive and very much still with us, are giving up center stage to the much smaller, more advanced microcomputers. Small, inexpensive computers need small inexpensive modems, so the old reliable 103-style modem is now even more popular than ever.

Baud Rates

The term baud often confuses people. And with good reason. It means different things depending on who is using the term, and when. In its narrowest technical definition, a baud is defined as being a measure of the rate at which signals are transmitted through a communications channel, with one baud corresponding to a rate of one signal element per second.

Well, that sounds pretty straightforward, but what's a signal element? With digital data it seems pretty reasonable that a signal element should be the same thing as a bit so that 300 baud would be the same thing as 300 bits per second. Right!? Well, sometimes.

Baud rate is concerned with the stuff (like tones) going down the communications channel (in this case a phone line), and NOT with the stuff (like ones and zeroes) we're turning it into at the other end. In a 103-style modem, each possible change from one frequency to the other is a signal element. Since that is how we code a single bit, one baud equals one bit per second. But the phone line has a very finite rate at which it can carry signal elements. The rate is related to the frequency range the line can accomodate and the modulation technique (as well as the error rate one is willing to tolerate). In general, 1200 baud is about the highest the phones will accomodate with any degree of reliability. So the fastest that a modem can be is 1200 baud (unless you're using some other kind of telephones).

Well where do 4800-baud modems come from then? Well, the answer is that they aren't 4800-baud modems at all. They are 4800 bit-per-second modems. They do it by very cleverly encoding 4 bits of digital data onto a single signal transition (corresponding to a single cycle of a 1200 Hz carrier tone). As you can imagine, it is quite a trick encoding them that way, and even more of one to get them back out at the other end. That's why they're so expensive -- it ain't easy.

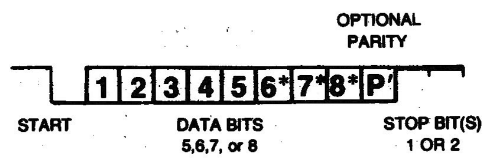
But that's not the only confusing part. As data communications users, you

and I don't care how many funny little bauds go down the phone line. In fact we really don't care about the bits even. We are trying to get some bytes moved around, but we'll probably settle for moving ASCII characters.

So how fast can I send characters with a 300 bit-per-second modem? Usually about 30 characters per second, but it can vary. You see, in order to be able to pick the characters out at the receiving end, we have to put on some extra bits to tell where the character begins and ends, etc. So our 7-bit ASCII character is accompanied by one start bit, one stop bit, an optional parity bit for error-checking, and an optional extra stop bit (I don't quite know why, it just slows things down). Thus our ASCII character could get from 2 to 4 extra bits as travelling companions.

The most common usage is one each of start bits, stop bits, and parity bits. This works out nicely because that means 10 bits per character which makes the character rate at 300 baud simple to calculate as 30 characters per second.

fig 8



Normally, there is no good reason to transmit any slower than you have to (you just get a bigger phone bill), except that it won't work if you send faster than whatever's at the other end can receive it. There are an awful lot of model 33 Teletypes still kicking around. They can send and receive only at 110 baud. Between all those TTY's and the machines built to talk to them, there are still a lot of 103-type modems connected to things that only run at 110 baud.

Half- and Full- Duplex

This is another very confusing aspect of data communications, and like baud rates is mostly confusing because the terms have been used so loosely and with so little regard for their original very narrow technical meaning.

A communications link which carries data from one point to another in only one direction is said to be a simplex link. Most of the devices (such as radio) which are used for communication are basically simplex devices. To get a two-way communication link, we have to use two simplex links, one going in each direction. This is called a duplex communications link. The telephone is a duplex communications link. The people at both ends of the phone can hear each other. They can even both talk at the same time (except on long-distance sometimes).

If we have a simplex link which can be turned around, then we can have two-way communications with only one communication path. This is called half-duplex, because it has the effect of a duplex link but with only half as much stuff. CB radio is half-duplex. When one person finishes talking, he has to say "over" or "10-4" or something so the person at he other end knows it's his turn to talk. If they both try to talk at the same time, neither one can hear anything.

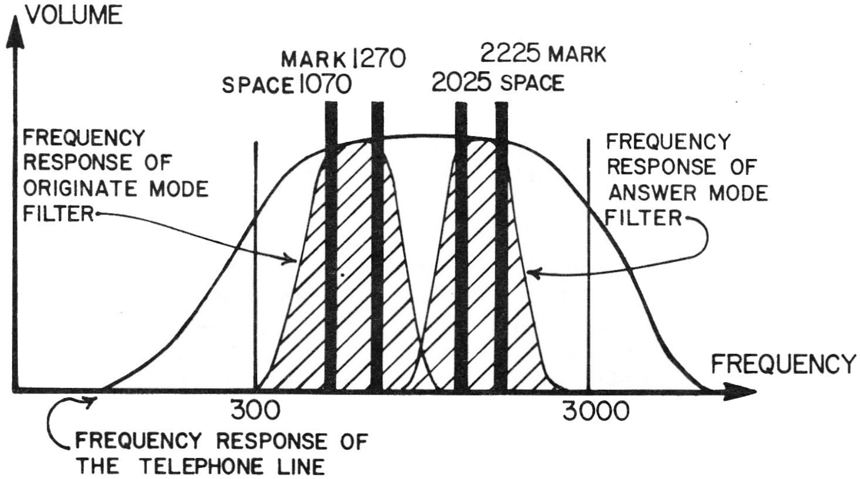
To have a full-duplex radio channel, each person would have to have a separate transmitter and receiver, and both people's transmitters would have to run all the time. In order for the two transmitters not to interfere with each other, they would have to use different channels. So full-duplex takes two channels.

Full-duplex communications is easier to use than half-duplex (at least for people), because it is more like normal face-to-face communication. So, even though it takes more stuff to make it work, the phones are built to work full-duplex. The phone in your house has a clever transformer called a hybrid which allows the two signal paths going in opposite directions to share the same pair of wires to the central office. But once it gets to the office, there are more circuits that sort them out, and they are kept that way until they leave another central office on their way to someone's phone.

Long-distance phone circuits always occur in pairs -- one circuit going in each direction. On calls that are over a few hundred miles, there is a problem with a full-duplex link. It takes the signals a finite length of time to get from one end to the other. When they arrive, part of the signal goes back into the phone and is sent back where it came from. You get an echo. To fix that, there are gizmos called echo suppressors in the lines. An echo suppressor is a voice-controlled switch that makes the phone line really work half-duplex, but it seems to be full-duplex because the echo suppressors turn around in the space between two syllables.

Well, the 103-type modem was designed to be able to take advantage of all those full-duplex communication paths. The designers of the 103 divided the frequency band of the telephone into two narrower bands, and assigned one for sending in one direction and the other for sending in the other. This gave them two channels to work with (like using two CB channels). It worked. Data could go in both directions at the same time. It worked on long distance, and the modems weren't at all bothered by the echos, because the modems were designed to talk on one frequency and listen on the other. They didn't care about the echos because they couldn't hear them.

fig 9

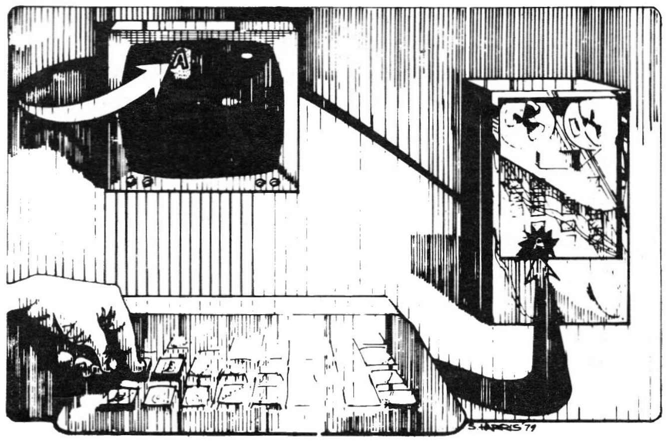


The only problem was those echo suppressors. They kind of messed things up. Fortunately, someone had already thought of that and put a special disable circuit into the echo suppressors, that made them turn off if they heard a tone in a certain band. One of the tones used by the 103 just happens to correspond to this signal. So as soon as an echo suppressor hears the carrier from a 103 modem, it turns itself off.

Not all modems are full-duplex. In fact, until recently the 103-type was the only full-duplex modem. Most of the faster modems are half-duplex. This makes them more complicated to use beause then the computers at both ends have to say "over" or "10-4" or something and all that. That, of course means a lot of complicated programming.

One nice thing about a full-duplex modem is that you can echo the characters. Virtually all time-sharing and data access systems echo each character back as it is received, and the character is not displayed or printed on the terminal until it has been for a full round trip to the distant computer and back. This gives the person typing a clear indication whether what he typed was received correctly. What he sees is what the computer sees. If a character gets garbled in the phone line, it shows up garbled on his terminal. If it gets lost completely, then nothing shows up on his terminal at all. This technique has a very catchy name, echo-plex.

fig 10



It is because of this practice that most terminals have a switch marked full- or half-duplex. If the terminal is connected to a half-duplex modem, echo-plex won't work because the computer can't send back the characters it receives. Some systems don't use echo-plex even when they can because it introduces a slight delay which some people find objectionable. In this case, the terminal must display the characters that it sends to the computer. This is all a full-/half-duplex switch does on any terminal. Often the same switch appears on modems (especially acoustic couplers) mostly because it's cheap and looks impressive.

Ringing and Dialing

Ringing and dialing are two aspects of what the telephone company calls signaling. Signaling is the process by which a connection is established on the switched (dial) telephone network. As is the case with many commonplace things, telephone signalling is much more complicated than it looks.

First let's consider what has to happen when you place a call. The dialog looks something like this:

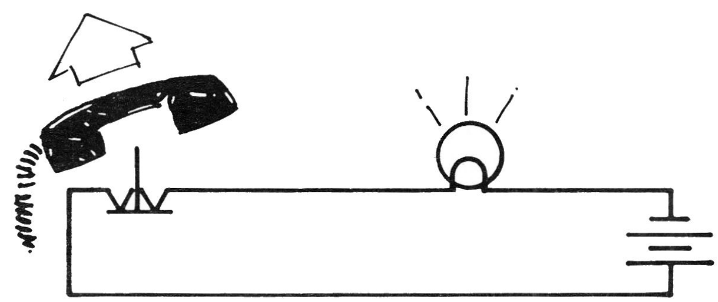
YOU	PHONE COMPANY
I want to make a call.	
	OK. Who do you want to call?
My girlfriend.	
	OK. I'll let her know you want to talk.
	I'm sorry, she's talking to someone else.

Now, that's not such a complicated dialog, but there are a few difficult limitations. For one thing, as we all know, there is seldom a person at the phone company, but some kind of a machine.

Everything that we say to the phone company has to be simple and exact enough for a machine, and everything the phone company says to us has to be simple enough that a machine can make the sounds. Whatever we do, we have to do it with just the two wires that connect us to the central office. And most limiting of all, we have to do it all with technology that was avaialable in 1894, because that's when the dial telephone was invented.

Let's start simple. How does the phone company find out that we've picked up the phone to make a call? Your telephone has a switch on which the receiver normally rests when not in use (the switch hook). When you lift the receiver, this switch establishes a connection between the two wires and allows current to flow from a battery at the central office. This current can be detected at the central office by a light bulb in series with the line, or by a relay.

fig 11

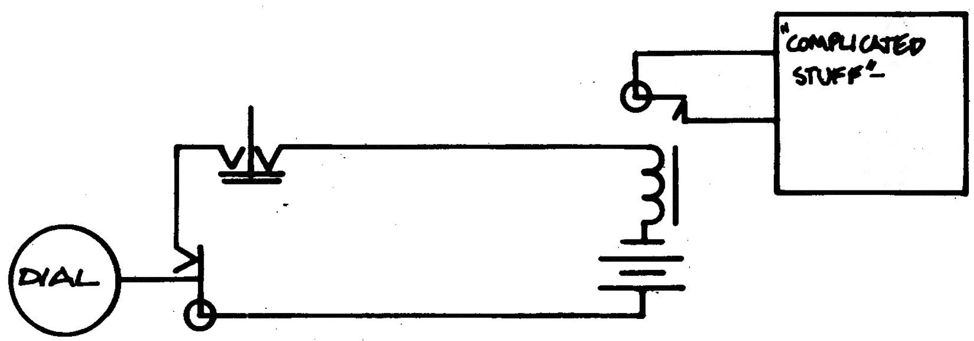


To tell you that it is listening, the central office machinery connects the line to a generator which produces an audible dial tone in your receiver.

To tell the machine who you want to talk to, you dial that person's telephone number. The dial in your phone has a switch that is in series with the switch hook. This switch momentarily breaks the connection between the two wires. When the connection breaks, it causes the relay at the central office to drop out momentarily.

The dial makes a series of these pulses very rapidly. The number of pulses corresponds to the digit you have dialed, from 1 to 10 pulses for the digits 1 to 0. At the central office these pulses are counted by Strowger relays and other more recent devices.

fig 12



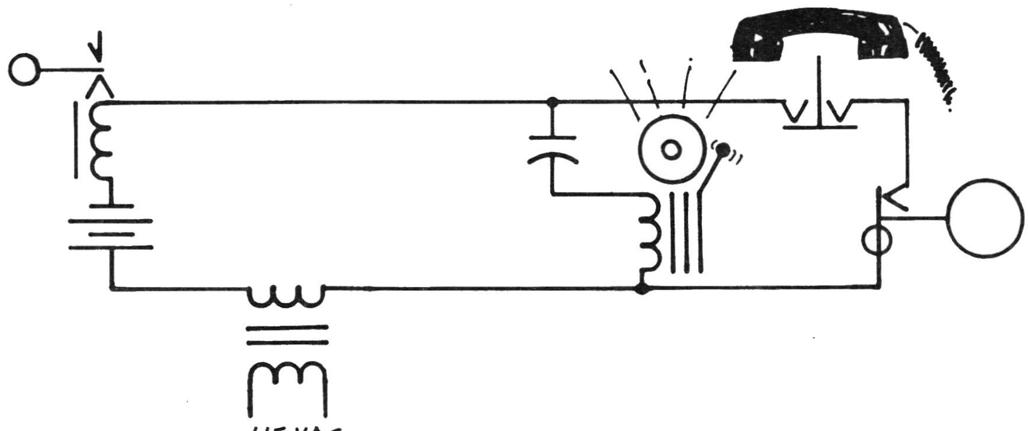
Once you have dialed, a whole bunch of complicated machines that I won't attempt to describe find the pair of wires corresponding to the number you have dialed.

If the line is busy, its phone will be off hook and the relay connected to it will be closed. The machine will then connect your line to another generator which makes a busy signal.

If the line is not busy, then it will connect that line to yet another generator. This one puts out 115 volts AC at 20 Hertz. Your friend's telephone has a bell which is connected to the line via a capacitor. The capacitor prevents the DC current from the battery from flowing through the bell, but allows the AC ring signal to pass through, thus making a loud noise.

When your friend picks up his phone, his switch hook makes connection, drawing current from the battery and activating the relay connected to his line. This causes the ring generator to be disconnected and stops the ringing.

fig 13



Over the years, many refinements and variations have been added, including tone dialing, in which tones are used instead of the current pulses I have described. Tone dialing may eventually replace pulse dialing, but now and for many years to come, all automatic telephone exchanges accept pulses even if they are designed for tone dialing. Many older exchanges are able to accept only pulse dialing, so for now, pulse dialing is the only universally usable technique.

Well, if you understand everything this far, you can consider yourself to be a minor expert on telephones and data communications. And as the owner of a Micromodem II, you are now equipped to use that knowledge to advance the state of the art by discovering and developing new applications. Good luck.

Specification for Micromodem II firmware

Entry Points

Name	Address	Function
ENTO	CN00	Initial input or output call from Apple II monitor. If MM2 is reset, Apply defaults. Determine by examination of console switch locations whether call is for input or output. Modify input or output console switch locations as necessary to direct further input and output to correct routines. Perform input or output function as deteremined above.
OUTA	CN02	Special output call for full duplex operation. Output a byte from location CHAR in a manner which avoids the 6502 false read problem.
OUT	CN05	Normal output call. Output byte in A reg subject to all output options etc.
IIN	CN07	Normal input location. Obtain a byte from either modem or keyboard in accordance with options set, and return it in the A reg.

Default Initialization

Default initialization is performed on entry via ENTO if the RESET signal is present from the modem. This signal is removed by operations performed by the initialization sequence and does not return unless a hardware reset occurs or someone clears the SET bit in the modem control register.

Default Settings

1)Lower-to-upper-case translation enabled.
2)KBDE FLAG bit set, all other FLAG bits reset.
3)Data format 8 data bits, no parity, 1 stop bit.
4)High baud rate (300).
5)Phone on hook, modem turned off.
6)30 msec line feed delay selected but not enable

Features of Input Routine (IIN)

1)If TERM flag set, enter TERMINAL mode.
2)If on-hook ignore input from modem.
3)If on-hook and phone rings, answer it:
    i)Display:

MICROMODEM II: RING

ii)At end of ring take phone off hook.
iii)Display:

MIICROMODEM II: AWAIT CARR.

iv)Put modem in answer mode.
v)Turn on carrier
vi)Wait up to 30 sec. for other carrier.
vii)If no carrier, abandon call.
viii)If carrier detected, display:

MICROMODEM II: CONN.

4)If TRAN flag not set, following control characters accepted from modem:

Character	Response
ctrl-T	Set TERM flag and enter TERMINAL mode.
ctrl-N	i)Set LFON flag to enable line feed insertion ii)Disable display (DISPO=1) iii)Clear Apple II screen.
ctrl-Y	Jump directly to Apple II monitor (FF65).
line feed	discard
null	discard
delete	discard

5)If TRAN flag not set and LOCSE = 20H then translate lower-case characters to equivalent upper-case.
6)If off-hook and no carrier is present, abandon call.
7)All ctrl-A sequences described in TERMINAL mode except ctrl-A ctrl-Q (start dialing) accepted from keyboard and processed regardless of status of KBDE (keyboard enable).
8)If KBDE flag set, accept all characters typed as valid input. If KBDE flag reset, ignore all keyboard input except ctrl-A sequences.
9)When call is abandoned:
i)Hang up phone.
ii)turn off modem.
iii)Display:

MICROMODEM II: NO CARR.
MICROMODEM II: HUNG UP

Features of Output Routine (OUT)

1)If on hook and character ctrl-Q sent, initiate DIALING sequence.
2)If dialing in progress, dial characters as sent (see DIALING).
3)If TRAN flag not set, and ctrl-Z sent, hang up phone, display:

MICROMODEM II: HUNG UP

4)If DISPO flag is zero (display selected), copy all modem output directly to the Apple II's display.
5)During actual transmission via the modem, the following characters are accepted from the modem input if TRAN is not set:

Character	Response
ctrl-S	Stop transmitting until any other character is received.
ctrl-Y	Jump directly to Apple II monitor (at FF65).

Features of DIALING

1)Initiated by output of ctrl-Q if on hook regardless of state of TRAN flag.
2)Initiated bya ctrl-A ctrl-Q in TERMINAL mode.
3)Once DIALING is initiated, it is not exited until a line feed, carriage return, or ctrl-Z character is detected.
4)On initiation, the following occurs:
  i)The following message is displayed:

MICROMODEM II: DIALING:

  ii)The phone is taken off hook.
  iii)A two-second delay is timed out to allow telephone exchange to return a dial tone.
5)The timing of dial pulses is in accordance with Bell System specifications:
  Each pulse = 61 msec on hook
  Interpulse delay = 39 msec
  Interdigit delay = 600 msec minimum
6)Characters output while DIALING are handled as follows:

Character	Response
Digits 1-9	Dialed as 1 to 9 pulses
Digit 0	Dialed as 10 pulses
*	2-second delay (for second dial tone)
{return}	i)Terminate dialing, ii)Set modem to originate mode, iii)Display:

MICROMODEM II: AWAIT CARR.

	iv)Wait up to 30 seconds for answering carrier, v)If carrier is detected, enable modem transmitter, vi)If no carrier detected, abandon call.
{line feed}	Terminate DIALING, do not enable modem or wait for response.
{ctrl-Z}	Abandon call.
all others	Display but otherwise ignore.

7)Ctrl-Z received from keyboard any time during 30-second wait for answering carrier causes call to be abandoned.

8)When call is abandoned, the following message is displayed:

MICROMODEM II: NO CARR.
MICROMODEM II: HUNG UP

Features of TERMINAL mode

1)Entered from input (IIN) when ctrl-A ctrl-F or ctrl-A ctrl-H sequences entered from keyboard.
2)Entered from input (IIN) if TERM flag set prior to calling IIN.
3)Exited (back to IIN) if ctrl-A ctrl-X entered from keyboard.
4)Exited (back to IIN) if ctrl-R received from modem and TRAN flag not set.
5)Entered if TRAN not set and ctrl-T received from modem.
6)On entry display message:

MICROMODEM II: BEGIN TERM

7)On exit display message:

MICROMODEM II: END TERM

8)Local display is switched through the console switch locations CSWL, CSWH and is compatible with Apple II serial and parallel printer interface cards.
9)The following control sequences are recognized from the keyboard.

Sequence	Response
ctrl-A ctrl-1	Set modem speed to 110 baud, format 8 data bits 2 stop bits.
ctrl-A ctrl-3	Set modem speed to 300 baud, format 8 data bits 1 stop bit.
ctrl-A ctrl-H	Set half duplex (enable display, DISPO=0)
ctrl-A ctrl-F	Set full duplex (disable display, DISPO=1)
ctrl-A ctrl-Z	Hang up phone and turn off modem.
ctrl-A ctrl-Q	Initiate dialing sequence.
ctrl-A ctrl-S	Transmit Break until any other character typed.
ctrl-A ctrl-X	Exit TERMINAL mode.

Subtle Points Not Covered Elsewhere

1)Shared ROM's are shut off by access to CFFF when entered from any of the four valid entry points.
2)Flashing cursor is removed from Apple II display when a character is received.
3)Flashing cursor is placed on screen when awaiting a character in TERMINAL mode.
4)Random number location (RNDH) incremented while awaiting characters.
5)X and Y registers and status of interrupt enable are preserved thru all entries.
6)ACIA control register (CR1) and modem control register (CR2) are refreshed from their RAM storage locations each time a character is transmitted.
7)The hex value CN (n=slot number) is maintained at loc 7F8 during all operations.

8)Shared ROM's are shut off after possible call to another peripheral card after characters displayed in TERMINAL mode.

Software-Controlled Options

There are many software-controlled options. This is an attempt merely to enumerate the most important ones. For operational details of individual options, please see the appropriate section of this manual.

Data Format

Two standard data formats may be selected via ctrl-A sequences from the keyboard: 8 bits, no parity, 2 stop bits at 110 baud, and 8 bits no parity, 1 stop bit at 300 baud. Other data formats may be selected by storing appropriate values in the RAM location for the ACIA register.

Baud Rate

300 baud is the default speed. 110 baud may be selected via a ctrl-A keyboard sequence. The rates may also be selected under software control by modifying the contents of the RAM location for the modem control register.

Lower-to-upper Case Translation

Normally enabled (20H in LOCSE), may be disabled by writing 0 to LOCSE.

Options Selected via Bits in FLAG Byte

Several software-controlled options are selected by setting or clearing appropriate bits in the FLAG byte in RAM.

Bit	Option Controlled
DISPO	Local display of data transmitted via modem.
LFI	Insertion of line feed after carriage return, also enables delay after return. Duration of delay is controlled by contents of location WAIT, which contains a delay factor in 10 msec units.
TERM	When set causes entry to TERMINAL mode from IIN.
TRAN	Code transparency. When set disables recognition of all control characters from any source with the following exceptions: i)ctrl-A sequences from keyboard. ii)ctrl-Q output if on hook iii){line feed} if LFON is selected

KBDE Keyboard Enable. If not set, no characters accepted from local keyboard except ctrl-A sequences.

TABLE OF MEMORY LOCATIONS

Note: In all addresses, N stands for the number of the slot in which the Micromodem II resides.

NO $6F8 = 1784 dec.

Contains the hex value NO any time the Micromodem II is in control computer.

CHAR $778 = 1912 dec.

Each character sent or received is temporarily stored here. Also used by self-test program, see SELFTEST (p. 8) for more information.

CN $7F8 = 2040 dec.

Contains the hex value CN any time the Micromodem II has control of the computer.

MODEM $678+N = 1656+N dec.

Contains a copy of the modem control byte. The actual hardware register is updated from this location each time a byte is transmitted. For further details see description for CR2 (p. 76).

FLAGS $778+N = 1912+N dec.

Contains 7 one-bit flags which control various functions of the Micromodem II firmware. The bits are:

bit no.	7	6	5	4	3	2	1	0
name	DISPO	DLS	X	LFI	TERM	TRAN	KBDE	DLG
weight	128	64	32	16	8	4	2	1

DISPO - When zero, causes all output to Micromodem II to be displayed on Apple II's display screen. THIS BIT ACTUALLY SETS THE DUPLEX MODE. WHEN IT'S ZERO HALF DUPLEX (LOCAL ECHO) IS SELECTED. WHEN IT'S SET TO A ONE FULL DUPLEX (NO LOCAL ECHO) IS SELECTED.

DLS - Used internally, indicates that Micromodem II firmware is preparing to start dialing.

X - Unused.

LFI - Enables line feed insertion after carriage return on output. Also enables delay after line feed, which is adjustable by setting location CRDLY.

TERM - Indicates that the terminal program is running. If this bit is set, the terminal program will begin running the next time the Micromodem II is polled for input.

TRAN - Transparent text mode. Causes Micromodem II firmware to ignore the usual control codes, except that control Q is still recognized if the Micromodem II is hung up. This is useful for some program-controlled applications.

KBDE - Keyboard enable. When this bit is set, the Micromodem II will accept input either from the Apple II's keyboard or from a remote keyboard connected via the telephone line. When this bit is reset, the Micromodem II will accept input only from the remote device, except for control-A sequences.

DLG - Used internally. Flag indicates that dialing is in progress.

<u>ACIA $7F8+N = 2040+N dec.</u>

Contains a copy of the ACIA control byte. The actual hardware register is updated from this location each time a byte is transmitted. For further details, see descrption of CR1 (p. 75).

<u>LOCSE $6F8+N = 1784+N dec.</u>

Upper/lower case translation flag. Normally contains $20 = decimal 32 which enables lower-to-upper case translation, or 0 to disable translation.

<u>DATA $C087+N0 = -16249+16*N dec.</u>

ACIA data input and output port. BASIC programs may read this port, but should not write to it. See description of self-test program (p. 8) for further details.

STATUS/CR1 $C086+N0 = -16250+16*N dec.

ACIA status and control ports. The following tables describe the bits used in the Micromodem II. For additional data please see the manufacturer's data sheet (Motorola MC6850). The status bits are:

bit no.	7	6	5	4	3	2	1	0
name	X	PE	OVRN	FE	RESET	$\overline{\text{CD}}$	TRE	RRF
weight	128	64	32	16	8	4	2	1

X - Unused.

PE - Parity error detected.

OVRN - Receiver overrun error.

FE - Framing error.

RESET - When set indicates that Apple II has been RESET since the last time the Micromodem II was initialized.

$\overline{\text{CD}}$ - Not carrier detect. When set indicates that no carrier is present, or carrier has been momentarily lost since the last data character was read.

TRE - Transmitter Register Empty. Indicates readiness of ACIA transmitter to accept another character.

RRF - Receiver Register Full. Indicates presence of a valid data character in receiver register.

The control bits are:

bit no.	7	6	5	4	3	2	1	0
name	0	0	0	LS3	LS2	LS1	0	1
weight	128	64	32	16	8	4	2	1

0 - Must always be 0.

LS3-LS1 - 3-bit character length select code. Please see ADVANCED PROGRAMMING (p. 37) for details.

1 - Must always be 1.

RI/CR2 $C085+N0 = -16251+16*N dec.

Modem status and control ports. On input, the most significant bit is 0 when the phone is ringing. The output bits are:

bit no.	7	6	5	4	3	2	1	0
name	OH	X	X	ST	SET	MODE	TXE	BRS
weight	128	64	32	16	8	4	2	1

OH - Off hook. When this bit is set, the Micromodem II "picks up the phone".

X - Unused.

ST - Self Test. When set causes Micromodem II to enter self-test mode. For details see description of self-test program (p. 8).

SET - Initialization flag. Used in conjunction with RESET status flag to control initialization of Micromodem II.
When this bit is set to zero, the RESET status flag goes to 1. When the Micromodem II is called for input or output immediately after an IN# or PR# statement, the RESET flag is checked, and if it is set, the default values are applied.

MODE - Selects originate or answer mode. 1 = originate.

TXE - Transmitter enable. Turns on modem transmitter when set.

BRS - Bit rate select. Selects high (300 baud) rate when set.

OUTA $C002+N00 = -16382+256*N

SPECIAL OUTPUT CALL LOCATION. A CALL OR JSR TO THIS LOCATION WILL CAUSE THE BYTE IN CHAR ($778 OR 1912 DEC.) TO BE OUTPUT TO THE MODEM.

MODIFYING THE DOW JONES STOCK REPORTER PACKAGE

REVISED 2/22/79

The Apple II Dow-Jones Stock Reporter Package is designed to run with the Apple Communications board. It requires some slight modifications to run with your Micromodem II, so that it will automatically dial the Dow-Jones computer and hang up when you are done.

This software package runs under Applesoft II, the Apple floating-point BASIC interpreter. Depending on the configuration of your machine, you may need to load this interpreter from tape or disk, or you may have it already loaded in ROM in your machine. In any case, the first step is to get Applesoft II running. You can recognize it by its prompt character ( ]).

Once you have the proper BASIC running, you will need to LOAD the Dow-Jones Stock Reporter package from the cassette on which it is supplied. Once you have it loaded into your computer, you can enter the following changes which will make the program run with the Micromodem II. Once you have made these changes, you should SAVE the program on a new tape so that you will not have to type in the changes again.

The table below shows the statements that need to be changed or added. If you are familiar with the Apple II's editing features, you may prefer to use them to save a little typing, otherwise it will probably be easier to simply retype the whole statements.

Line 20 should contain the telephone number of your nearest Dow-Jones access port, which you can obtain from the Dow-Jones manual with your stock reporter package.

Some of the lines will be longer than the screen is wide. This will cause them to fold over into two or more lines. This will not bother the computer, so you should not let it bother you either.

TABLE OF CHANGES

```
LINE     CHANGE

20       DIM NUMBER$(15) : NUMBER$ = "your local dow-jones access number"
1004     PR#CSLOT : PRINT : POKE 1912+CSLOT,128 : PR#0
1330     PR#CSLOT : PRINT CHR$(17); NUMBER$ : PR#0
1340     IF PEEK(1656+CSLOT) > 127 THEN GOTO 1360
1350     INPUT "NO ANSWER, PRESS RETURN TO TRY AGAIN";A$
1355     GOTO 1330
1630     PR#CSLOT : PRINT CHR$(26) : PR#0
2275     PRINT " 1. CHANGE A STOCK"
2292     PRINT " 5. HANG UP PHONE"
2295     PRINT:PRINT "TYPE 1,2,3,4 OR 5 ";
2310     IF I<1 OR I>5 THEN 2294
2320     ON I GOTO 3000,11000,12000,2350,2325
2325     PR#CSLOT:PRINT CHR$(26) : PR#0
2327     LGIN=0 : GOTO 2235
10045    PR#CSLOT : PRINT CHR$(26) : PR#0
```

Once you have entered all the changes you should use the LIST command to check your work. Simply type:

LIST <line number>

Replacing <line number> with the line number you are checking.

When you are sure that all the changes have been made correctly, you should use the SAVE command to save your work on another tape (or on disk).

Procedures and Information for using Datamover With the MICROMODEM II

First it is necessary to install a patch to your copy of Datamover.

1) Load the Datamover. It resides at locations 800 thru A4F hex.
2) Start the Apple mini-assembler (F666G, see page 69 in Apple II Reference Manual).
3) Enter the following lines:

```
897:LDX 7F8
 LDA #A
 STA 6B8,X
 JSR FDOC
 JMP FF65
```

4) Save the patched program on tape or disk for later use.

The Datamover program and its operation are documented in the manual Communications Interface Card Addendum to the Installation and Operating Manual published by Apple Computer, Inc. When using the Micromodem II with this program, at the points where you are instructed to put the phone in the acoustic coupler, simply type {CTRL}A {CTRL}Q {phone number}{RETURN} and the Micromodem II will establish the necessary telephone connection.

Example Session Using Datamover to Load and Run the Telepong Game Program, With D.O.S.

In this example, it is assumed that the patches to Datamover have already been made and the results BSAVE'd in a file called DATAMOVER, and that the Telepong program has been SAVE'd in a file called TELEPONG. Both computers are assumed to be equipped with a Micromodem II in slot 3. Computer 1 is assumed to have a Disk II in slot 6. Computer 2 may otherwise be a minimal Apple II configuration.

Data to be typed by the operators is in lower case, and upper case is used to denote data printed by the computers. {return}'s are not indicated, but are needed at some places. This should be obvious in most cases.

Computer 1	Computer 2
{reset}	{reset}
*6{ctrl}p	*{ctrl}b
>print "{ctrl}d in#3"	>in#3
>print "{ctrl}d pr#0"	
>load telepong	
>{ctrl}a	
MICROMODEM II:?{ctrl}h	
{ctrl}a	
MICROMODEM II:?{ctrl}q	
MICROMODEM II:DIALING {phone number}	
MICROMODEM II:AWAITING CARR.	MICROMODEM II:RING
MICROMODEM II:CONNECT	MICROMODEM II:CONNECT
{ctrl}a	
MICROMODEM II:? {ctrl}x	
{return}	
>brun datamover	
(HEX DISPLAY OF DATAMOVER PGM)	(HEX DISPLAY OF DATAMOVER PGM)
("NOTHING HAPPENS" FOR ABOUT 30 SECONDS)	("NOTHING HAPPENS" FOR ABOUT 30 SECONDS)
FINISHED	FINISHED
MICROMODEM II:BEGIN TERM	MICROMODEM II:BEGIN TERM
{ctrl}r	MICROMODEM II:END TERM
	>
{ctrl}a	
MICROMODEM II:? {ctrl}x	
run	
COMMUNICATION CARD SLOT? 3	
	COMMUNICATION CARD SLOT? 3
(THE GAME RUNS)	(THE GAME RUNS)
(END OF GAME)	(END OF GAME)
{BEEP}	{BEEP}
>	>

(both operators wish to play another game)

Computer 1	Computer 2
	in#3 (note that this must come first)
run	

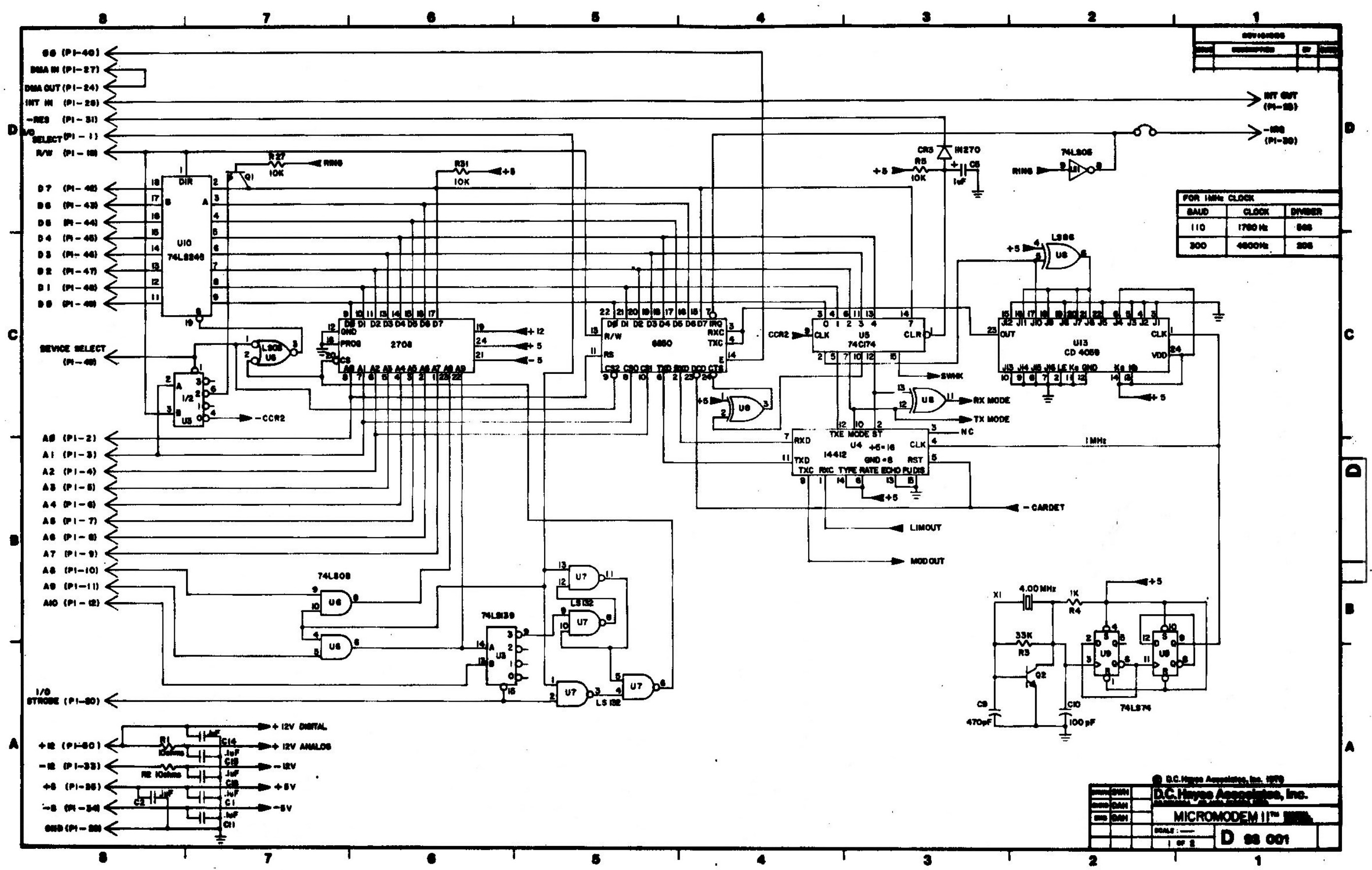
(ANOTHER GAME)

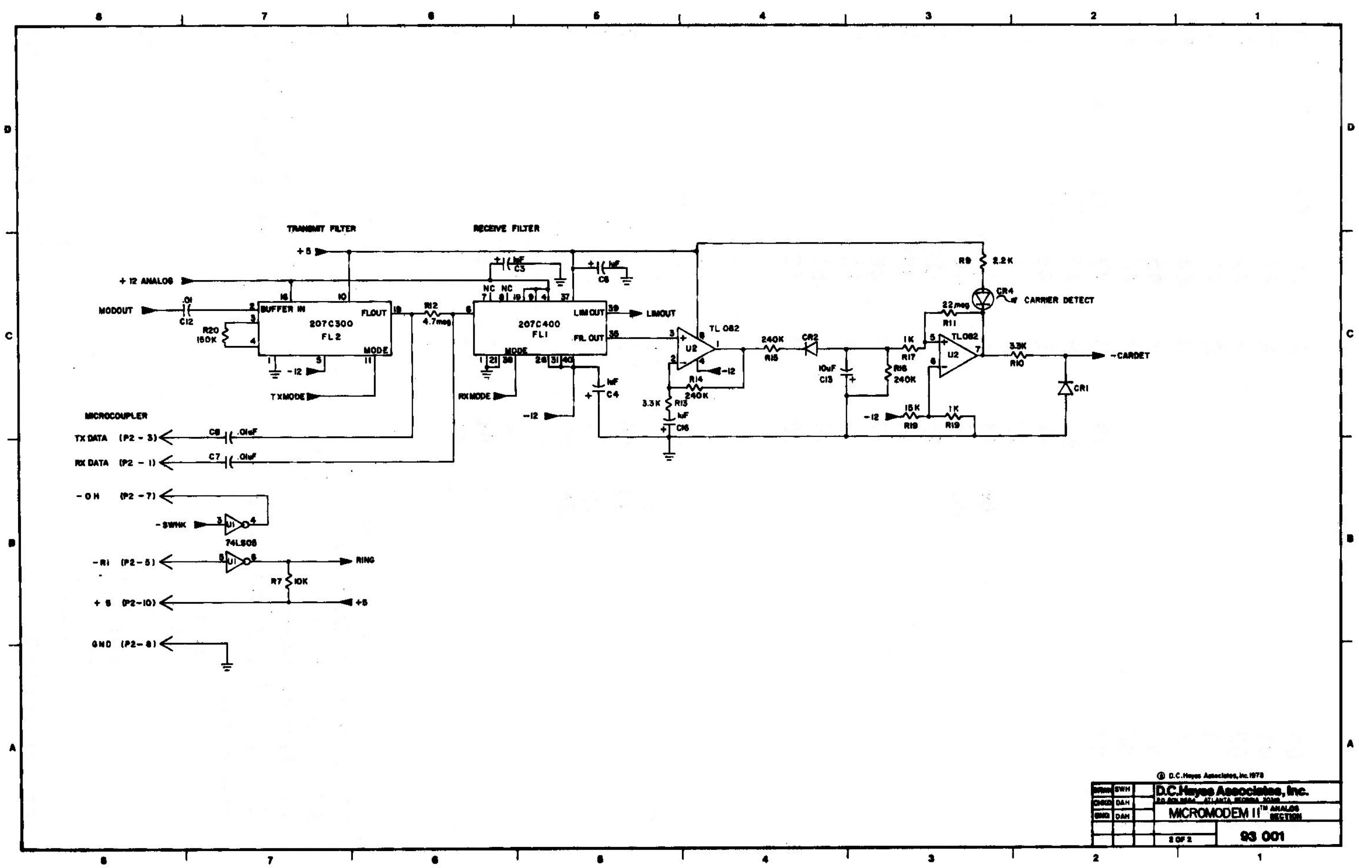
(operators wish to communicate with each other)

Computer 1	Computer 2
in#3	in#3

```
{ctrl}a                                    {ctrl}a
MICROMODEM II:? {ctrl}h                    MICROMODEM II:? {ctrl}h
MICROMODEM II:BEGIN TERM                   MICROMODEM II:BEGIN TERM
        (operators converse by typing to each other)

                ...
                ...
        (THIS COULD GO ON INDEFINITELY)
```

S0 (P1-40)
DMA IN (P1-27)
DMA OUT (P1-24)
INT IN (P1-28)
-RES (P1-31)
I/O SELECT (P1-1)
R/W (P1-18)
D7 (P1-42)
D6 (P1-43)
D5 (P1-44)
D4 (P1-45)
D3 (P1-46)
D2 (P1-47)
D1 (P1-48)
D0 (P1-49)
SERVICE SELECT (P1-41)
A0 (P1-2)
A1 (P1-3)
A2 (P1-4)
A3 (P1-5)
A4 (P1-6)
A5 (P1-7)
A6 (P1-8)
A7 (P1-9)
A8 (P1-10)
A9 (P1-11)
A10 (P1-12)
I/O STROBE (P1-20)
+12 (P1-50)
-12 (P1-33)
+5 (P1-25)
-5 (P1-34)
GND (P1-26)
+12V DIGITAL
+12V ANALOG
-12V
+5V
-5V
U10 74LS245
2708
6850
U5 74C174
U13 CD 4059
U4 14412
74LS08
74LS139
LS132
LS02
LS86
74LS05
74LS74
R27 10K
R31 10K
R5 10K
CR3 IN270
RING
-CCR2
CCR2
SWHK
RX MODE
TX MODE
NC
1MHz
-CARDET
LIMOUT
MODOUT
4.00 MHz
X1
1K
R4
33K
R3
Q2
C9 470pF
C10 100 pF
INT OUT (P1-23)
-IRQ (P1-30)
FOR 1MHz CLOCK
BAUD CLOCK DIVIDER
110 1760 Hz 568
300 4800Hz 208
© D.C. Hayes Associates, Inc. 1979
D.C. Hayes Associates, Inc.
MICROMODEM II™
1 of 2
D 98 001


TRANSMIT FILTER
RECEIVE FILTER
+5
+12 ANALOG
MODOUT
C12
R20 150K
BUFFER IN
207C300
FL2
FLOUT
MODE
TXMODE
-12
R12 4.7meg
207C400
FL1
LIM OUT
FIL OUT
RXMODE
LIMOUT
C3
C6
C4
TL 082
U2
R14 240K
R13 3.3K
C16
240K R15
CR2
10uF C13
R16 240K
1K R17
22meg R11
TL082
R9 2.2K
CR4
CARRIER DETECT
3.3K R10
-CARDET
CR1
15K R18
1K R19
MICROCOUPLER
TX DATA (P2 - 3)
RX DATA (P2 - 1)
C8 .01uF
C7 .01uF
- O H (P2 - 7)
-SWHK
74LS05
U1
- RI (P2 - 5)
RING
R7 10K
+ 5 (P2 - 10)
GND (P2 - 8)
© D.C. Hayes Associates, Inc. 1978
D.C. Hayes Associates, Inc.
MICROMODEM II™ ANALOG SECTION
SWH
DAH
DAH
2 OF 2
93 001

ASCII CODE TABLE

American Standard Code for Information Interchange

CODE	HEX	DEC	CODE	HEX	DEC	CODE	HEX	DEC	CODE	HEX	DEC
NUL	00	0	SP	20	32	@	40	64	`	60	96
SOH	01	1	!	21	33	A	41	65	a	61	97
STX	02	2	"	22	34	B	42	66	b	62	98
ETX	03	3	#	23	35	C	43	67	c	63	99
EOT	04	4	$	24	36	D	44	68	d	64	100
ENQ	05	5	%	25	37	E	45	69	e	65	101
ACK	06	6	&	26	38	F	46	70	f	66	102
BEL	07	7	'	27	39	G	47	71	g	67	103
BS	08	8	(	28	40	H	48	72	h	68	104
HT	09	9	)	29	41	I	49	73	i	69	105
LF	0A	10	*	2A	42	J	4A	74	j	6A	106
VT	0B	11	+	2B	43	K	4B	75	k	6B	107
FF	0C	12	,	2C	44	L	4C	76	l	6C	108
CR	0D	13	-	2D	45	M	4D	77	m	6D	109
SO	0E	14	.	2E	46	N	4E	78	n	6E	110
SI	0F	15	/	2F	47	O	4F	79	o	6F	111
DLE	10	16	0	30	48	P	50	80	p	70	112
DC1	11	17	1	31	49	Q	51	81	q	71	113
DC2	12	18	2	32	50	R	52	82	r	72	114
DC3	13	19	3	33	51	S	53	83	s	73	115
DC4	14	20	4	34	52	T	54	84	t	74	116
NAK	15	21	5	35	53	U	55	85	u	75	117
SYN	16	22	6	36	54	V	56	86	v	76	118
ETB	17	23	7	37	55	W	57	87	w	77	119
CAN	18	24	8	38	56	X	58	88	x	78	120
EM	19	25	9	39	57	Y	59	89	y	79	121
SUB	1A	26	:	3A	58	Z	5A	90	z	7A	122
ESC	1B	27	;	3B	59	[	5B	91	{	7B	123
FS	1C	28	<	3C	60	\	5C	92	|	7C	124
GS	1D	29	=	3D	61	]	5D	93	}	7D	125
RS	1E	30	>	3E	62	^	5E	94	~	7E	126
US	1F	31	?	3F	63	_	5F	95	DEL	7F	127

NUL Null, or all zeros
SOH Start of Heading
STX Start of Text
ETX End of Text
EOT End of Transmission
ENQ Enquiry
ACK Acknowledge
BEL Bell, or Alarm
BS Backspace
HT Horizontal Tab
LF Line Feed
VT Vertical Tab
FF Form Feed
CR Carriage Return
SO Shift Out
SI Shift In
DLE Data Link Escape
DC1 Device Control 1
DC2 Device Control 2
DC3 Device Control 3
DC4 Device Control 4
NAK Negative Acknowledge
SYN Sync
ETB End Transmission Block
CAN Cancel
EM End of Medium
SUB Substitute
ESC Escape
FS File Separator
GS Group Separator
RS Record Separator
US Unit Separator
SP Space
DEL Delete

D.C. Hayes Associates, Inc.

16 PERIMETER PARK DR. SUITE 101
P.O. BOX 9884 ATLANTA, GEORGIA, 30319 (404) 455-7663